stichting
mathematisch
centrum

MC



AFDELING MATHEMATISCHE BESLISKUNDE BN 14/72 AUGUSTUS

B.J. LAGEWEG, J.K. LENSTRA

# ALGORITMEN VOOR KNAPZAKPROBLEMEN

Voorlopige uitgave

2e boerhaavestraat 49 amsterdam

*Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.*

*The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O), by the Municipality of Amsterdam, by the University of Amsterdam, by the Free University at Amsterdam, and by industries.*

I N H O U D

## 1. INLEIDING

We beschouwen het volgende probleem:

$$P(b) \qquad \max \sum_{j=1}^{n} c_j x_j$$

$$(1) \qquad \text{onder} \sum_{j=1}^{n} a_j x_j \leq b$$

$$0 \leq x_j \leq u_j \qquad j = 1,\ldots,n$$

$$x_j \text{ geheel} \qquad j = 1,\ldots,n$$

Dit probleem zullen we aanduiden als het rugzakprobleem met bovengrenzen. We onderscheiden twee speciale versies van dit probleem. Als alle bovengrenzen $u_j = 1$ zijn, spreken we over het knapzakprobleem KP(b) met knapzakfunctie K(a), $0 \leq a \leq b$:

$$KP(b) \qquad K(b) = \max \sum_{j=1}^{n} c_j x_j$$

$$(2) \qquad \text{onder} \sum a_j x_j \leq b$$

$$x_j = 0,1 \qquad j = 1,\ldots,n$$

Het tweede geval, het rugzakprobleem, treedt op als de bovengrenzen geen beperking vormen, d.w.z. $u_j \geq b/a_j$.

De rugzakfunctie is gedefinieerd als:

$$RP(b) \qquad R(b) = \max \sum_{j=1}^{n} c_j x_j$$

$$(3) \qquad \text{onder} \sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \geq 0, \text{ geheel} \qquad j = 1,\ldots,n$$

We nemen aan dat $a_j$ $(j=1,\ldots,n)$ en b natuurlijke getallen zijn en dat $c_j$ $(j=1,\ldots,n)$ niet-negatief en reëel is.
We gebruiken in het vervolg de notaties:

$H_{(p,q)}(b)$ is de maximale waarde van probleem HP(b), met $x_1 = \ldots = x_{p-1} = 0$ en $x_{q+1} = \ldots = x_n = 0$;

$H_{-d}(b)$ is de maximale waarde van probleem HP(b), met $x_d = 0$;

$[y]$ is het grootste gehele getal niet groter dan y;

$d_j \equiv c_j/a_j$ is de relatieve waardedichtheid van variabele $x_j$.

Als toepassingen vermelden we het investeringsselectieprobleem [18] en het snijprobleem [7], waarin de rugzakalgoritme een subroutine is van het L.P.-probleem. Aan de meer theoretische kant is te noemen:

- Het groepenprobleem, dat ontstaat bij de groepentheoretische aanpak van het geheeltallige programmeringsprobleem [18], kan worden geformuleerd als een rugzakprobleem met bovengrenzen.
- Bij de reductie van stelsels vergelijkingen in geheeltallige variabelen ontstaat een knapzakprobleem, waarin de bijvoorwaarde een gelijkheid is [2] .
- De sneden in snede-algoritmen voor het geheeltallige programmeringsprobleem kunnen worden versterkt door het oplossen van een rugzakprobleem [1].

Twee oplossingsmethoden worden behandeld:
dynamische programmering in hoofdstuk 2 en branch-and-bound in hoofdstuk 3.
Geen aandacht wordt gegeven aan knapzakalgoritmen, die de toegelaten hoekpunten van de eenheidskubus genereren [3] [13].

## 2. DYNAMISCHE PROGRAMMERING

### 2.1 De rugzakalgoritmen van Hu en Nemhauser

Uitgangspunt voor toepassing van dynamische programmering is het model:

$$(4)\quad \begin{cases} F_0(a) \equiv 0 \\ F_j(a) \equiv \max\limits_{0 \leq x_j \leq [a/a_j]} \{c_j x_j + F_{j-1}(a-a_j x_j)\}, & j=1,\ldots,n \end{cases}$$

$$(5)\quad z_j(a) \equiv x_j, \text{ als } F_j(a) = c_j x_j + F_{j-1}(a-a_j x_j), \quad j=1,\ldots,n, \; a=0,1,\ldots,b$$

Door herhaalde substitutie van deze definitie volgt $F_j(a) = R_{(1,j)}(a)$ en in het bijzonder $F_n(b) = R_{(1,n)}(b) = R(b)$.
Een oplossing die $F_n(b)$ realiseert wordt recursief teruggevonden als:

$$(6)\quad x_j = z_j\,(b - \sum_{l=j+1}^{n} a_l x_l), \qquad j=n,n-1,\ldots,1.$$

Het model (4) - (5) heeft twee, voor practisch gebruik onoverkomenlijke, bezwaren: Het aantal te vergelijken beslissingen is zeer groot - bij benadering $nb + \frac{1}{2}b^2 \sum_{j=1}^{n} 1/a_j$ - en de functies $z_1,\ldots,z_n$ moeten voor $0 \leq a \leq b$ worden bewaard om de optimale strategie terug te vinden.
Een iets andere aanpak (Hu [12]) komt aan deze bezwaren tegemoet.
We definiëren:

$$(7)\quad \begin{cases} H_0(a) \equiv \begin{cases} -\infty & a < 0 \\ 0 & a \geq 0 \end{cases} \\ H_j(a) \equiv \max\{H_{j-1}(a),\ c_j + H_j(a-a_j)\}, & j=1,\ldots,n \end{cases}$$

$$(8)\quad \begin{cases} i_0(a) \equiv 0 \\ i_j(a) \equiv \begin{cases} j & \text{als } H_j(a) > H_{j-1}(a) \\ i_{j-1}(a) & \text{anders} \end{cases}, & j=1,\ldots,n. \end{cases}$$

Lemma 1: $H_j(a) = R_{(1,j)}(a)$ voor $0 \leq a \leq b$ en $1 \leq j \leq n$.

Bewijs: Als $a < 2a_j$ zijn de definities (4) en (7) identiek.
Als $pa_j \leq a < (p+1)a_j$, $p \geq 2$ volgt door substitutie:

$$H_j(a) = \max \{H_{j-1}(a),\ c_j + \max \{H_{j-1}(a-a_j),\ c_j + H_j(a-2a_j)\}\}$$

$$= \max \{H_{j-1}(a),\ c_j + H_{j-1}(a-a_j), \ldots, pc_j + H_{j-1}(a-pa_j)\}$$

$$\equiv F_j(a) = R_{(1,j)}(a).$$

Om een oplossing die $H_n(b)$ realiseert terug te vinden, is alleen $i_n(a)$, $a=0,\ldots,b$ nodig. Stel $a_0 \equiv 0$ en bepaal recursief de indexverzameling

$$I(b) \equiv \{j_1 = i_n(b),\ j_2 = i_n(b-a_{j_1}), \ldots, j_m = i_n(b-a_{j_1} - \ldots - a_{j_{m-1}}), \ldots\}.$$

De waarde van $x_j$ in een optimale oplossing van RP(b) is de cardinaliteit van j in I(b).
Het aantal te vergelijken beslissingen bij deze methode is bij benadering nb. De registratie van de optimale oplossing vergt slechts één array met lengte b.

De algoritme van Hu (procedure 1 van appendix C) is gebaseerd op de functionaalvergelijkingen (7) en (8). Vooraf wordt nagegaan of bepaalde eenvoudige vormen van dominantie (zie appendix A) optreden. De daarna resterende variabelen worden gesorteerd naar niet-stijgende relatieve dichtheden. De berekening (7) van de rugzakfunctie voor variabele j wordt slechts uitgevoerd, indien $c_j > R_{(1,j-1)}(a_j)$.

De functies $H_j(a)$ zijn monotoon niet-dalende trapfuncties, die in principe alleen op de sprongpunten berekend behoeven te worden (Nemhauser [18] ). De algoritme van Nemhauser (procedure 2 van appendix C) is in opbouw gelijk aan de algoritme van Hu, met dien verstande dat alleen sprongpunten geadministreerd worden.

## 2.2 Rugzakalgoritmen van Gilmore & Gomory

In de voorgaande algoritmen is de rekentijd evenredig met het produkt van het aantal variabelen n en het rechterlid b van de bijvoorwaarde. In principe staan dus twee wegen open om tot effectievere algoritmen te komen, waarvoor wij gebruikmaken van specifieke eigenschappen van de rugzakfunctie.

Lemma 2: Voor de r.z.f $R(a)$, $0 \leq a \leq b$ geldt:

(9) $R(a) \geq 0$

(10) $R(a_j) \geq c_j$ , $j=1,\ldots,n$

(11) $R(a) \geq R(a')$ , $a > a'$

(12) $R(a+a') \geq R(a) + R(a')$

Bewijs (12): De som van de optimale oplossingen van $RP(a)$ en $RP(a')$ is een toegelaten, maar niet noodzakelijk optimale oplossing van $RP(a+a')$.

Lemma 3: Als $x_p > 0$ in een oplossing die $R(a)$ realiseert, dan:

(13) $R(a) = R(a-a_p) + R(a_p)$

(14) $R(a_p) = c_p$

Bewijs: Zij $R(a) = \sum_{j=1}^{n} c_j x_j$, $x_p > 0$. Een toegelaten oplossing voor $RP(a-a_p)$ wordt gevormd door

$$x_j^* = \begin{cases} x_j & , j \neq p \\ x_p - 1 & , j = p. \end{cases}$$

Uit lemma 2 volgt nu (13):

$$R(a) \geq R(a-a_p) + R(a_p) \geq \sum_{j=1}^{n} c_j x_j^* + c_p = R(a).$$

Door (13) als definitie van $R(a_p)$ te beschouwen, volgt (14):

$$c_p \leq R(a_p) \leq \sum_{j=1}^{n} c_j x_j - \sum_{j=1}^{n} c_j x_j^* = c_p .$$

Gevolg 1:(Optimaliteit van substrategiën)

Als $(x_1,\ldots,x_n)$ R(b) realiseert,
realiseert $(y_1,\ldots,y_n)$, met $0 \leq y_j \leq x_j$ ,
R(a) voor $\sum_{j=1}^{n} a_j y_j \leq a \leq \sum_{j=1}^{n} a_j y_j + (b - \sum_{j=1}^{n} a_j x_j)$.

Gevolg 2: De r.z.f R(a) voldoet aan de functionaalvergelijking:

$$(15) \qquad \begin{cases} R(0) = 0 \\ R(a) = \max \{R(a-a_j) + c_j) \mid 0 \leq j \leq n, \ a_j \leq a\} \text{ , voor } a > 0, \end{cases}$$

waarbij de verschilvariabele $x_0$ gedefinieerd is door $a_0=1$ en $c_0 = 0$.

De algoritme 1 A van Gilmore & Gomory [9] (procedure 3 van appendix C) is gebaseerd op functionaalvergelijking (15), geschreven als:

(16) $R(a) = \max \{R(y) + c_j \mid y = a - a_j, \ a_j \leq a, \ 0 \leq j \leq n\}$ , $a > 0$.

Er zijn twee arrays met lengte b in gebruik: array r bevat na afloop de rugzakfunctie R, in array i staat op plaats a een beslissing j, waarvoor $R(a) = R(a-a_j) + c_j$.

De algoritme exploiteert het feit dat alleen sprongpunten van de r.z.f R(a) in aanmerking behoeven te worden genomen als y in (16). Als voor $y = a - a_k$, $k > 0$ geldt: $R(y) = R(y-1)$, dan volgt:
$R(y) + c_k = R(y-1) + c_k + c_0 \leq R(y + a_k - 1) + c_0 = R(a-1) + c_0$,
zodat de term voor $j = k$ in het rechterlid van (16) gemajoreerd wordt door de term voor $j = 0$.

We nemen aan dat voor alle variabelen geldt $0 < a_j \leq b$ en $a_j \neq a_k$ voor $j \neq k$, en dat de variabelen geïndiceerd zijn naar stijgende $a_j$.

Algoritme 1 A luidt:

0. Initialiseer $r(a) = 0$ voor $0 \leq a \leq b$.
   Initialiseer $r(a_j) = c_j$ en $i(a_j) = j$ voor $j = 1,\ldots,n$.
   Stel $i(0) = 0$ en $y = 1$.

1. Als $r(y) \leq r(y-1)$, kies dan als optimale beslissing $i(y) = 0$ en stel $r(y) = r(y-1)$.

2. (Controleer op dominantie).

3. Als $i(y) = 0$, ga dan naar 5.
4. Bereken voor $a = y + a_j$ , $j = 1,\ldots,n$ met $a_j \leq \min(y, b-y)$, $r(y)+c_j$. Als $r(y) + c_j > r(a)$, stel dan $r(a) = r(y) + c_j$ en $i(a) = j$.
5. Als $y < b$, verhoog dan y met 1 en ga naar stap 1.
6. Bereken de oplossing, die r(b) realiseert door backtracking in array i vanuit $a = \max \{ y \mid 0 \leq y \leq b,\ i(y) > 0 \}$. Stop.

Opmerkingen:

1. Dominantie van variabele j kan worden geconstateerd in stap 2 als $y = a_j$. Als $i(y) \neq j$, bestaat een combinatie van variabelen, waarin j niet voorkomt, met $r(a_j) \geq c_j$: elimineer variabele j.
2. In stap 3 mag gelden: $a_j \leq y$, om vergelijkingen tussen oplossingen, die bij backtracking de volgorde $j,\ldots,i(y)$, resp. $i(y),\ldots,j$ zouden opleveren, te vermijden.
3. Afgezien van de beperking tot sprongpunten van R(a) is algoritme 1 A wat betreft aantal vergelijkingen ($n \times b$) en benodigde geheugenruimte gelijkwaardig aan de algoritme van Hu.

Een functiewaarde R(a) wordt mogelijk door verschillende oplossingen gerealiseerd. We willen aan a een unieke oplossing x(a) toekennen bij een gegeven permutatie $(0,1,\ldots,n)$ van de variabelen $x_0, x_1, \ldots, x_n$. We kiezen als deze unieke x(a) de lexicografisch grootste oplossing die R(a) realiseert: als $x \neq x(a)$ eveneens R(a) realiseert, bestaat een getal p met $x_j = x_j(a)$ voor $j < p$ en $x_p < x_p(a)$. We definiëren voorts de functie i(a) als

$$
\begin{aligned}
& i(0) \equiv n+1 \\
& i(a) \equiv \min \{ j \mid 0 \leq j \leq n,\ R(a) = R(a-a_j) + c_j \}, \quad a > 0.
\end{aligned} \tag{17}
$$

i(a) wijst de variabele met de laagste index aan, die positief is in een of andere oplossing die R(a) realiseert (lemma 3), m.a.w.
$i(a) = \min \{ j \mid x_j(a) > 0 \}$.
Als we backtracken in array i, met $i(a) = j$, vinden we derhalve alleen indices, die niet kleiner zijn dan j.
Want stel $i(a-a_j) = p < j$. In de unieke oplossing $x(a-a_j)$ is dan $x_p(a-a_j) > 0$. Maar $x(a-a_j)$, aangevuld met eenmaal variabele j, realiseert R(a) en is lexicografisch groter dan x(a), in strijd met de definitie van x(a).

We hebben aangetoond:

Lemma 4 [9] : Als $i(a) = j$, dan $i(a-a_j) \geq j$ voor $a > 0$.

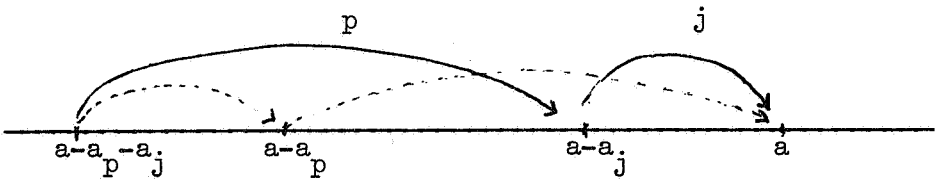
Gevolg 3: De r.z.f. $R(a)$ voldoet aan:

$$\begin{cases} R(0) = 0 \\ R(a) = \max \{R(a-a_j) + c_j \mid 0 \leq j \leq i(a-a_j),\ a_j \leq a\}, & a > 0, \end{cases}$$

waarbij $i$ gedefinieerd is door (17).

Het aantal vergelijkingen volgens (18) is minder dan volgens (15). Een verdere reductie kunnen we bereiken door op te merken dat een beslissing alleen optimaal kan zijn, als hij ook optimaal was in een "eerdere vergelijkbare situatie":

Lemma 5 [20]: Als $i(a) = j$ en $i(a-a_j) = p \geq j$, dan $i(a-a_p) = j$.




Bewijs: Triviaal voor $p = j$. Neem aan $p > j$. In $x(a)$ zijn $x_j(a)$ en $x_p(a)$ positief. Uit lemma 3 volgt:

$$R(a) = c_j + c_p + R(a-a_j-a_p) = c_p + R(a-a_p),$$

ofwel $R(a-a_p) = c_j + R(a-a_j-a_p)$.

Uit de definitie (17) van de i-functie volgt $i(a-a_p) \leq j$.

Stel $i(a-a_p) = q < j$.

$$R(a) = c_p + R(a-a_p) = c_p + c_q + R(a-a_p-a_q) \leq c_q + R(a-a_q),$$

m.a.w. $R(a)$ wordt gerealiseerd door een oplossing, waarin $x_q > 0$, $q < j = i(a)$, in strijd met de definitie van $i(a)$.

Gevolg 4: De r.z.f. $R(a)$ voldoet aan:

(19)
$$\begin{cases} R(0) = 0 \\ R(a) = \max \{R(a-a_j)+c_j \mid 0 \leq j \leq i(a-a_j),\ j = i(a-a_{i(a-a_j)}),\ a_j \leq a\}, & a>0, \end{cases}$$

waarbij i gedefinieerd is door (17).

Opmerking: Bij toepassing van gevolg 4 moet $i(a-a_{i(a-a_j)})$ bekend zijn in $a-a_j$, d.w.z. $a_j \leq a_{i(a-a_j)}$ voor $j \leq i(a-a_j)$:

de variabelen moeten naar stijgende $a_j$ worden geïndiceerd.

Behalve door vermindering van het aantal te vergelijken beslissingen, kan ook winst geboekt worden door verkleining van het interval, waarover R(a) berekend moet worden. Als $i(a) = 1$ voor $a' \leq a \leq a''$, met $a'' - a' \geq \max_j a_j$, komen voor $a > a''$ alleen nog beslissingen $x_1$ (en $x_0$) in aanmerking (lemma 4). De vraag is of en hoe we naar zo'n situatie kunnen toewerken. Intuïtief verwacht men dat variabelen met een grote relatieve dichtheid "op den duur" de voorkeur genieten. Deze verwachting wordt gepreciseerd in

Lemma 6:

Als voor $x_p$ geldt: $d_p \geq d_j$ voor alle j, $p < j \leq n$, dan bestaat een getal a(p), zó, dat voor alle $a \geq a(p)$:

(20) $R(a) = R_{(p+1,n)}(a') + R_{(0,p)}(a'')$,

voor een $a' \leq a(p)$ en $a' + a'' \leq a$.

Bewijs: Zij $v_j$ en $w_j$ geheel, met $v_j a_j = w_j a_p$ voor $j = p+1, \ldots, n$.
We kiezen $a(p) = \sum_{j=p+1}^{n} v_j a_j$. R(a) kan met herhaald toepassen van lemma 3 gesplitst worden in twee stukken, met alle variabelen $x_j$, $j > p$ in het eerste stuk:

$R(a) = R_{(p+1,n)}(a') + R_{(0,p)}(a'')$.

We moeten nog aantonen dat $a' \leq a(p)$. Als $a' > a(p)$, is er een variabele $x_q$, $q > p$, met $x_q > v_q$. Vervang $x_q$ door $x_q' = x_q - v_q$ en $x_p$ door $x_p' = x_p + w_q$. Dit reductieproces voor $x_q > v_q$ kan worden herhaald, totdat alle $x_q \leq v_q$. Bijgevolg is tenslotte $a' \leq a(p)$.

Gevolg 5: Als voor $x_1$ geldt $d_1 \geq d_j$, $1 < j \leq n$ ($x_1$ is een "turnpike" variabele), dan voldoet R(a) voor $a > a(1)$ aan:

$R(a) = R_{(2,n)}(a') + [(a-a')/a_1]c_1$ voor een $a' \leq a(1)$,

$R(a) = R(a-a_1) + c_1$.

R(a) is voor a > a(1) periodiek met periodelengte $a_1$ en een niveauverschil $c_1$ tussen twee opeenvolgende perioden. Als we een turnpike variabele index 1 geven, zijn op den duur alleen nog beslissingen $x_1$ toegelaten. Zodra dat het geval is - gewoonlijk voor een $a(1) << \sum_{j=2}^{n} v_j a_j$ -, kan R(a) verder door extrapolatie worden gevonden.

Algoritme 1 B van Gilmore & Gomory [9] / Shapiro & Wagner [20] (procedure 4 van appendix C) is gebaseerd op functionaalvergelijking (19) en de periodiciteit van de rugzakfunctie. De structuur van de algoritme is identiek aan algoritme 1 A. In stap 4 echter worden alleen beslissingen met $j \leq i(y)$ en $j = i(a-a_{i(y)})$ beschouwd. In stap 5 wordt gestopt, zodra een a bereikt is, waarboven de rugzakfunctie periodiek wordt, d.w.z. als voor alle y > a in stap 4 alleen beslissing 0 of 1 is toegelaten.
We nemen aan dat variabele 1 een turnpike variabele is en dat de overige variabelen geïndiceerd zijn naar stijgende $a_j$. De algoritme luidt:

0. Initialiseer $r(a) = i(a) = 0$ voor $0 \leq a \leq b$.
   Initialiseer $r(a_j) = c_j$ en $i(a_j) = j$ voor $j = 1,\ldots,n$.
   Stel $i(0) = n+1$ en $y = 1$.
1. Als $r(y) \leq r(y-1)$, kies dan als optimale beslissing
   $i(y) = 0$ en stel $r(y) = r(y-1)$.
2. (Controleer op dominantie).
3. Als $i(y) = 0$, ga dan naar 5.
4. Bereken voor $a = y + a_j$ met $j = 1,\ldots,i(y)$, $a \leq b$ en $j = i(a-a_{i(y)})$, $r(y) + c_j$. Als $r(y) + c_j > r(a)$, stel dan $r(a) = r(y) + c_j$ en $i(a) = j$.
5. Bepaal $bmax = \max \{a \mid i(a) > 1\}$.
   Als $y < bmax$, verhoog dan y met 1 en ga naar stap 1.
6. Bepaal de oplossing, die r(b) realiseert, door extrapolatie voor $b > bmax + a_1$ en backtracking voor het resterende gedeelte. Stop.

## 2.3 Knapzakalgoritmen

De knapzakfunctie voldoet aan de eigenschappen (9) - (11) van de rugzakfunctie: $K(a) \geq 0$, $K(a_j) \geq c_j$ en $K(a) \geq K(a')$ als $a \geq a'$, maar niet aan eigenschap (12). Als $x(a)$, resp. $x(a')$ de functiewaarden $K(a)$, resp. $K(a')$ realiseren, dan hoeft $x(a) + x(a')$ geen toegelaten oplossing van $KP(a + a')$ te zijn. Het is dus niet mogelijk knapzakalgoritmen af te leiden, analoog aan de rugzakalgoritmen van Gilmore & Gomory.
We kunnen $K(b)$ als volgt recursief uitdrukken:

$$K_0(y) \equiv 0 \quad , \quad 0 \leq y \leq b$$
$$i_0(y) \equiv 0 \quad , \quad 0 \leq y \leq b$$

$$(21) \qquad K_j(y) \equiv \begin{cases} K_{j-1}(y) & , \ 0 \leq y < a_j \\ \max\{K_{j-1}(y),\ K_{j-1}(y-a_j) + c_j\} & , \ a_j \leq y < b \end{cases}$$

$$i_j(y) \equiv \begin{cases} 1 & , \ K_j(y) > K_{j-1}(y) \\ 0 & , \ \text{anders} \end{cases} \quad , \ 0 \leq y \leq b$$

$$, \ 1 \leq j \leq n.$$

$K(b)$ is gelijk aan $K_n(b)$ en de bijbehorende oplossing wordt recursief bepaald als: $x_j = i_j(b - \sum_{p=j+1}^{n} a_p x_p)$, $j = n, n-1, \ldots, 1$.

In tegenstelling tot het rugzakprobleem zijn nu alle n functies $i_j$ nodig om de oplossing terug te vinden.
De knapzakalgoritme van Hu (procedure 5 van appendix C) is gebaseerd op de functionaalvergelijking (21). De variabelen worden gesorteerd op niet-stijgende relatieve dichtheid, omdat wijziging van de functiewaarde in $k(y)$ meer handelingen vereist dan een gelijkblijvende functiewaarde. De knapzakfunctie $K_{(1,j)}(a)$ wordt berekend over het interval $[0, \min\{b, \sum_{l=1}^{j} a_l\}]$.

De algoritme luidt:

0. Sorteer de variabelen op niet-stijgende $c_j/a_j$.
1. Stel $k(0) = 0$, $bmax = 0$ en $j = 1$.
2. Als $bmax < b$, bepaal dan $b' = \min\{bmax + a_j, b\}$, initialiseer $k(y) = k(bmax)$ voor $y = bmax + 1, \ldots, b'$ en stel $bmax = b'$.

3. Bepaal achtereenvolgens voor y = bmax, bmax - 1,...,$a_j$ de waarde $k(y-a_j) + c_j$. Als $k(y-a_j) + c_j > k(y)$ stel dan $k(y) = k(y-a_j) + c_j$.
4. Leg in array i de omslagpunten van de functie $i_j(y)$, gedefinieerd volgens (21) vast, d.w.z. alle y waarvoor $i_j(y) \neq i_j(y+1)$.
5. Als $j < n$ verhoog dan j met 1 en ga naar 2.
6. Als bmax < b, stel dan b = bmax.
7. Bepaal de oplossing die k(b) realiseert door backtracking in array i middels telling van het aantal omslagpunten van de functie $i_j(y)$ op het interval $[b, b - \sum_{l=j+1}^{n} a_l x_l]$.
8. Stop.

Gerhardt [6] ondervangt het bezwaar dat alle n functies $i_j$ bewaard moeten worden door een andere definitie van $i_j$:

$$(22)\qquad \begin{aligned} i_j(y) &\equiv \begin{cases} j & \text{, als } K_j(y) > K_{j-1}(y) \\ i_{j-1}(y) & \text{, anders} \end{cases} \qquad \text{, } j \geq 1 \\ i_0(y) &\equiv 0. \end{aligned}$$

De oplossing van KP(b) wordt bepaald door backtracking in $i_n(y)$, zolang de daarbij aangetroffen indices van de variabelen monotoon dalen. Immers, zij $j_1 = i_n(b)$, $b_2 = b - a_{j_1}$ en $j_2 = i_n(b_2)$. Als $j_2 < j_1$, dan $i_{j_2}(b_2) = i_n(b_2)$. Als echter $j_2 \geq j_1$, dan $K_n(b_2) > K_{j_1-1}(b_2)$ en $i_n(b_2) = j_2 \neq i_{j_1-1}(b_2)$. In dat geval moeten $K_{j_1-1}(b_2)$ en $i_{j_1-1}(b_2)$ opnieuw berekend worden door het knapzakprobleem $KP(b_2)$ voor $j_1-1$ variabelen op te lossen. Het voordeel dat minder geheugenruimte nodig is kan het nadeel meebrengen, dat een - moeilijk voorspelbaar - aantal keren een knapzakprobleem moet worden opgelost. De knapzakalgoritme van Gerhardt (procedure 6 van appendix C) is in opzet gelijk aan de algoritme van Hu. De verschillen betreffen de initialisering in stap 2, de registratie van de functie $i_j(y)$ in stap 4 en het backtrackproces in stap 7. Deze stappen luiden voor de algoritme van Gerhardt:

2. Als bmax < b, bepaal dan b' = min $\{bmax + a_j, b\}$ ,
initialiseer k(y) = k(bmax) en i(y) = i(bmax) voor y = bmax,...,b'
en stel bmax = b'.

4. Stel i(y) = j, als in stap 3 de waarde van k(y) veranderd is,
voor y = bmax, bmax-1,...,$a_j$.

7a. Stel n = i(b), $x_n$ = 1 en verminder b met $a_n$.

7b. Als k(b) = 0, ga dan naar 8.

7c. Als i(b) $\geq$ n, verminder dan n met 1 en ga naar 1.

7d. Ga naar 7a.

## 3. BRANCH-AND-BOUND

### 3.1 Algemeen

We houden ons bezig met het algemene probleem P(b). We nemen in het vervolg aan dat de variabelen geïndiceerd zijn naar niet-stijgende relatieve dichtheden, d.w.z. $c_j/a_j \geq c_k/a_k$ als $j < k$, en dat $0 < a_j \leq b$ voor alle variabelen.

Zij $X$ de verzameling van alle toegelaten oplossingen van P(b). We bekijken deelverzamelingen $X$ die worden gekarakteriseerd door een vector met n componenten $\bar{x} = (\bar{x}_1,\ldots,\bar{x}_n)$, met $\bar{x}_j \geq -1$ en geheel. Een variabele $j$ is in $X$ gefixeerd op de waarde $\bar{x}_j$ als $\bar{x}_j \geq 0$, en vrij in $X$ als $\bar{x}_j = -1$; we noteren $X$ wel als $X = \{(\bar{x}_1,\ldots,\bar{x}_n)\}$. De vector $\bar{x}$ bepaalt bij $X$ een indexverzameling van vrije variabelen $N(X) \equiv \{j \mid \bar{x}_j = -1\}$, en twee getallen: de som $c(X)$ van de waarden van de gefixeerde variabelen:

$$c(X) = \sum_{j \in N \setminus N(X)} c_j \bar{x}_j \quad ,$$

en het herziene rechterlid $b(X)$:

$$b(X) = b - \sum_{j \in N \setminus N(X)} a_j \bar{x}_j .$$

Een bovengrens $UB(X)$ van $X$, in de zin van de branch-and-boundtheorie [17], kan worden berekend door de eis van geheeltalligheid van de variabelen te verwaarlozen:

$$UB(X) = \max \sum_{j=1}^{n} c_j x_j .$$

$$\text{onder} \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$0 \leq x_j \leq u_j \quad , \; j \in N(X)$$

$$x_j = \bar{x}_j \quad , \; j \in N \setminus N(X),$$

ofwel:

$$(23)\quad UB(X) = c(X) + \max \{ \sum_{j \in N(X)} c_j x_j \mid \sum_{j \in N(X)} a_j x_j \leq b(X),\ 0 \leq x_j \leq u_j,\ j \in N(X) \}.$$

De optimale oplossing van dit L.P.-probleem is eenvoudig te bepalen. De variabelen $x_j, j\in N(X)$, krijgen de waarde $u_j$ in volgorde van index, zolang $\sum_{\substack{l\in N(X)\\ l\leq j}} a_l u_l \leq b(X)$. De variabele met de laagste index, zeg $x_f$, waarvoor het linkerlid $b(X)$ overtreft, krijgt waarde $(b(X) - \sum_{\substack{l<f\\ l\in N(X)}} a_l u_l)/ a_f$; de overige variabelen $x_j$, $j\in N(X)$ worden 0 gesteld.

We definiëren op het branch-and-bound proces de ondergrensfunctie LB als de waarde van de beste tot dusver bekende oplossing van P(b), die we in het vervolg de pretendent zullen noemen.
Een verzameling X heet gepeild, als $UB(X) \leq LB$. Een niet-lege deelverzameling kan gepeild worden.

1. door de optimale oplossing van X te bepalen; als de waarde hiervan groter is dan LB, registreer dan deze oplossing als pretendent en stel LB gelijk aan deze waarde;
2. door rechtstreekse berekening van een bovengrens $UB(X)\leq LB$;
3. door splitsing van X in disjuncte deelverzamelingen $X_1, X_2, \ldots$ en peiling van deze deelverzamelingen:

$$UB(X) \leq \max_{p\geq 1} UB(X_p) \leq LB.$$

Als de eerste manier van peilen te moeilijk is - voor is dit het eigenlijke probleem - en rechtstreekse berekening van UB(X) niet tot resultaat leidt, wordt het splitsingsprincipe te hulp geroepen:
X wordt indirect gepeild door peiling van een aantal disjuncte deelverzamelingen, in de hoop dat deze verzamelingen gemakkelijker rechtstreeks gepeild kunnen worden. De peiling van deze deelverzamelingen kan onmiddellijk na de splitsing dan wel op een later tijdstip worden uitgevoerd (zie kanttekening 2 in par.3.4).
Probleem P(b) is equivalent met het peilen van , waarbij na afloop LB de waarde heeft van de optimale oplossing van P(b), die gegeven wordt door de pretendent.

## 3.2 Rugzakalgoritme

In het rugzakprobleem kunnen de bovengrenzen op de variabelen als oneindig groot worden beschouwd. De bovengrens van een deelverzameling X is dan

$$UB(X) = \begin{cases} c(X) & , \ N(X) = \emptyset \\ c(X) + d_k b(X) & , \ N(X) \neq \emptyset, k = \min \{j \mid j \in N(X)\} \end{cases} .$$

X wordt zonodig gesplitst in deelverzamelingen $X_0,\ldots,X_M$ door de in X vrije variabele met de laagste index, de splitsingsvariabele, zeg $x_k$, in $X_p$ te fixeren op waarde p, d.w.z.

$$X_p \equiv \{(\bar{x}_1,\ldots,\bar{x}_{k-1}, p, \bar{x}_{k+1},\ldots, \bar{x}_n)\} \ , \text{ voor } p=0,1,\ldots,M=[b(X)/a_k].$$

De nieuw gegenereerde deelverzamelingen $X_0,\ldots,X_M$ kunnen als volgt gepeild worden:

1. Als er in $X_M$ nog vrije variabelen een gehele positieve waarde kunnen aannemen, stel dan p = M en ga naar 3.
2. De in $X_M$ optimale oplossing $x^*$ is $x_j^* = \begin{cases} \bar{x}_j & j \notin N(X) \\ 0 & j \in N(X) \end{cases}$ ,
   met waarde $c(X_M)$. Als $c(X_M) > LB$, stel dan $LB = c(X_M)$ en registreer $x^*$ als pretendent.
   Als $UB(X_M) = c(X_M)$, d.w.z. als $X_M$ geen vrije variabelen meer heeft of $b(X_M) = 0$, is $X_M$ (en impliciet $X_{M-1},\ldots,X_0$) gepeild; ga dan naar 8.
   Vervolg anders met peiling van de eerstvolgende niet-lege deelverzameling $X_p$, d.i. stel $p = [(b(X) - \min_{j \in N(X)} a_j)/a_k]$ en ga naar 7.
3. Bereken $UB(X_p)$. Als $UB(X_p) \leq LB$ bevat $X_p$ (en a fortiori $X_{p-1},\ldots,X_0$) geen optimale oplossing: ga naar 8.
4. Bereken de vrije variabele $x_s$ met de kleinste index die in $X_p$ waarde 1 kan aannemen: $s \equiv \min \{j \mid j \in N(X_p), a_j \leq b(X_p)\}$ .
   Als $s > k+1$, kunnen $x_{k+1},\ldots,x_{s-1}$ in $X_p$ alleen waarde nul hebben: verander in dat geval de karakteristieke vector van $X_p$ in $(\bar{x}_1,\ldots,\bar{x}_k,0,\ldots,0, \bar{x}_s,\ldots,\bar{x}_n)$, verscherp de bovengrens tot $UB(X_p) = c(X_p) + d_s b(X_p)$ en ga naar stap 6 als deze bovengrens $UB(X_p) \leq LB$.

5. Splits $X_p$ in $1 + [b(X_p)/a_s]$ deelverzamelingen door $x_s$ als splitsingsvariabele voor $X_p$ te nemen en peil deze nieuw gegenereerde groep deelverzamelingen.

6. $X_p$ is gepeild, verminder p met 1.

7. Als $p \geq 0$, ga dan naar 3.

8. Stop: $X_M,\ldots,X_0$ zijn gepeild.

De rugzakalgoritme (procedure 7 van appendix C) [7] initialiseert LB=0, kiest $x_1$ als splitsingsvariabele van (deel)verzameling en peilt $0,\ldots,[b/a_1]$.

Opmerking: In procedure 7 worden LB en c(X) niet afzonderlijk berekend, maar alleen het verschil LB - c(X).

## 3.3 Knapzakalgoritme

We kunnen bij knapzakproblemen een deelverzameling $X \equiv \{(\bar{x}_1,\ldots,\bar{x}_k, -1,\ldots,-1)\}$ splitsen in $X_1 = \{x \mid x\epsilon X, x_s = 1, s > k\}$ en $X_0 = \{x \mid x\epsilon X, x_s=0, s > k\}$, analoog aan de rugzakalgoritme.
Een andere manier om X te splitsen is in deelverzamelingen, die onderscheiden worden door de in X vrije variabele met de laagste index die in de desbetreffende deelverzameling op waarde 1 gefixeerd is. Als we definiëren:

$$X^p \equiv \{(\bar{x}_1,\ldots,\bar{x}_k, 0,\ldots,0, \bar{x}_p = 1,-1,\ldots,-1)\},\ p=k+1,\ldots,n$$

$$X^{n+1} = \{(\bar{x}_1,\ldots,\bar{x}_k, 0,\ldots,0)\},$$

dan is $X = \bigcup_{p=k+1}^{n+1} X^p$ en $X^p \cap X^q = \emptyset,\ p \neq q$.

Als we zo splitsen in $^1,\ldots,^{n+1}$ en nieuw gegenereerde deelverzamelingen telkens verder splitsen in volgorde van stijgende index, is het resulterende proces een lexicografische aftelling van oplossingen van KP(b). Omdat aftelling van een deelverzameling slechts voortgezet behoeft te worden, zolang deze niet gepeild is, kan de aftelling voor een groot deel impliciet geschieden.

De bovengrens UB(X) op $X = \{(\bar{x}_1,\ldots,\bar{x}_k=1,-1,\ldots,-1)\}$ is volgens (23):

$$UB(X) = c(X) + \sum_{l=k+1}^{f-1} c_l + d_f \{b(X) - \sum_{l=k+1}^{f-1} a_l\} \tag{24}$$

$$\equiv c'(X) + d_f \cdot b'(X)$$

Omdat berekening van deze bovengrens meer werk is dan bij het rugzak-probleem, is het zaak deze berekening zo weinig mogelijk uit te voeren en een eenmaal berekende bovengrens zoveel mogelijk te benutten:

a. Als $a_p > b(X)$, dan $X^p = \emptyset$.

b. Als $X^{k+1} = \ldots = X^n = \emptyset$, dan is $(\bar{x}_1,\ldots,\bar{x}_k, 0,\ldots,0) \in X^{n+1}$ optimaal in X en is X rechtstreeks gepeild.

c. $UB(X^p)$ is een monotoon niet-stijgende functie van p.
Als $X^p$ gepeild is, zijn ook $X^{p+1},\ldots,X^{n+1}$ gepeild, overeenkomstig de definitie van gepeilde deelverzameling.

d.1 Als $X^{k+1} \neq \emptyset$, dan $UB(X^{k+1}) = UB(X)$.

d.2 Als na splitsing $X^q$ de eerste niet-lege deelverzameling is, $X^{k+1} = \ldots = X^{q-1} = \emptyset$ en $X^q \neq \emptyset$, dan geldt:

$$UB(X^q) \leq c'(X) + d_q\, b'(X),$$

$$\text{en} \quad UB(X^q) = c'(X) + \sum_{l=q}^{s-1} c_l + d_s b'(X^q),$$

$$\text{als} \quad UB(X^q) = c(X^q) + \sum_{l=q+1}^{s-1} c_l + d_s b'(X^q).$$

De bovengrens van de "eerste" niet-lege deelverzameling van een nieuw gegenereerde groep deelverzamelingen kan dus overschat of berekend worden met behulp van de bovengrens van de moederverzameling.

e. Binnen een groep deelverzamelingen kunnen schattingen en bovengrenzen op soortgelijke wijze doorgegeven worden tussen twee na elkaar onderzochte deelverzamelingen.

e.1 Als $(X^p)^{p+1} \neq \emptyset$, d.w.z. in $X^p$ kan variabele $x_{p+1}$ waarde 1 aannemen, dan ook $X^{p+1} \neq \emptyset$. De bovengrensfunctie daalt van $X^p$ naar $X^{p+1}$ met tenminste $c_p - d_f a_p$. Een eenvoudige overschatting van $UB(X^{p+1})$, die vooral kracht heeft als $f >> p$, is

$$UB(X^{p+1}) \leq UB(X^p) - c_p + d_f a_p = \{c'(X^p) - c_p\} + d_f\{b'(X^p) + b_p\}.$$

$UB(X^{p+1})$, die volgens (24) gevonden wordt als

$$UB(X^{p+1}) \equiv c'(X^{p+1}) + d_s\, b'(X^{p+1}),$$

kan ook via enkele correcties op $UB(X^p)$ bepaald worden:

$$UB(X^{p+1}) = \{c'(X^p) - c_p\} + \sum_{l=f}^{s-1} c_l + d_s b'(X^{p+1}).$$

e.2 Als $(X^p)^{p+1} = \emptyset$, $X^{p+1} = \ldots = X^{q-1} = \emptyset$ en $X^q \neq \emptyset$, dan is een overschatting voor $UB(X^q)$:

$$UB(X^q) \leq UB(X^p) - c_p + d_q.a_p$$
$$= \{c'(X^p) - c_p\} + d_q\{b'(X^p) + a_p\},$$

en kan $UB(X^q)$ worden berekend als

$$UB(X^q) = \{c'(X^p) - c_p\} + \sum_{l=q}^{s-1} a_l + d_s b'(X^q).$$

Onder de veronderstelling dat minstens één vrije variabele in $X = \{(\bar{x}_1,\ldots \bar{x}_k, -1,\ldots,-1)\}$ waarde 1 kan aannemen, d.w.z. $\bigcup_{p=k+1}^{n} X^p \neq \emptyset$, kunnen we X als volgt "aftellen tot peiling":

1. Als $X^{k+1} \neq \emptyset$, stel dan p = k+1 en ga naar 5.
   Initialiseer p = k+1.
2. Bepaal de eerstvolgende niet-lege, nog niet gepeilde deelverzameling $X^p$, d.w.z. stel $p = \min \{l \mid a_l \leq b(X), l > p\}$.
3. Bereken $UB(X^p)$ met behulp van d. of e. en stel f gelijk aan de index van de fractionele variabele in (24).
   Als $UB(X^p) \leq LB$, is $X^p$ (en a fortiori $X^{p+1},\ldots,X^n$) gepeild: ga naar 8.
4. Als de oplossing, die $UB(X^p)$ realiseert, geheeltallig is, i.c. $UB(X^p) = c'(X^p)$, registreer dan deze oplossing $(\bar{x}_1,\ldots,\bar{x}_k,0,\ldots,0,\bar{x}_p = 1, 1, \ldots,\bar{x}_{f-1} = 1,0,\ldots,0)$ als pretendent, stel $LB = c'(X^p)$ en ga naar 8.
5. Als er in $X^p$ nog vrije variabelen waarde 1 kunnen aannemen - hetzij $f > p+1$, hetzij $\min \{a_j \mid j > f\} \leq b(X^p)$ -, pas dan het aftelproces toe op $X^p$ en ga naar 2.
6. Als $c'(X^p) > LB$, registreer dan de oplossing, die $c'(X^p)$ realiseert, als pretendent en stel $LB = c'(X^p)$.
7. Als $\bigcup_{l=p+1}^{n} X^l \neq \emptyset$, i.c. $\min \{a_j \mid j > p\} > b(X^p) + a_p$, ga dan naar 2.
8. Stop: $X^{p+1},\ldots,X^n$ zijn gepeild.

De knapzakalgoritme (procedure 8 van appendix C) [7] [12] initialiseert LB = 0 en peilt door impliciete aftelling.

* Als de overschatting voor $UB(X^p)$, gebaseerd op d. of e. hierboven, LB niet overtreft, ga dan naar 8.

## 3.4 Kanttekeningen

1. In bovenstaande algoritme kan als splitsingsvariabele worden beschouwd de vrije variabele met de laagste index. Greenberg en Hegerich [11] splitsen op de fractionele variabele in een deelverzameling, d.i. de variabele die bij oplossing van het L.P.probleem (23) ter bepaling van de bovengrens een niet-gehele waarde aanneemt. Hun algoritme (procedure 9 van appendix C) is equivalent aan de algoritme van Land and Doig [15] voor het algemene geheeltallige lineaire programmeringsprobleem, als deze algoritme wordt gebruikt om KP(b) op te lossen.

2. Alle boven beschreven algoritmen onderzoeken gegenereerde deelverzamelingen volgens de LIFO-regel : de laatst gegenereerde deelverzameling wordt het eerst gepeild. Voordelen van deze aanpak zijn een eenvoudige, automatische administratie van nog te peilen deelverzamelingen en een gering aantal handelingen tussen twee opeenvolgend onderzochte deelverzamelingen. Nadeel is de starheid, waardoor eventueel meer deelverzamelingen gegenereerd worden dan strikt noodzakelijk, door splitsing van deelverzamelingen met een bovengrens, die kleiner is dan de waarde van de optimale oplossing.
   Dit bezwaar vervalt, als we een prioriteitsregel gebruiken, waarbij we steeds proberen de nog niet-gesplitste deelverzameling met de hoogste bovengrens te peilen.
   Terwijl i.h.a. een compromis tussen de LIFO-regel en een prioriteitsregel wordt aanbevolen [5], is onze ervaring, in overeenstemming met Greenberg en Hegerich [11], dat voor knapzakproblemen de LIFO-regel de voorkeur verdient.

3. Procedure 10 van appendix C (rucksackubbb) is een algoritme voor het rugzakprobleem met bovengrenzen.
   De algoritme is met inachtneming van de bovengrenzen op de variabelen opgebouwd als procedure 7 voor het rugzakprobleem zonder bovengrenzen.

4. Cabot [4] gebruikt de eliminatiemethode van Fourier-Motzkin om branch-and-bound algoritmen voor het rugzakprobleem af te leiden. In deze algoritmen kunnen de splitsingsvariabelen in elke willekeurige, zij het vaste volgorde, worden gekozen. Wordt gesplitst naar dalende

relatieve dichtheid, dan resulteert de boven beschreven rugzakalgoritme. Cabot rapporteert de beste resultaten bij splitsing naar stijgende range van de variabelen, waarbij hij de range bepaalt als het verschil tussen de maximale en minimale waarde van een variabele in een toegelaten oplossing, die een hogere kriteriumwaarde heeft dan een heuristisch bepaalde beginoplossing.

## 4. REKENRESULTATEN

De ALGOL-procedures 1 - 10 (appendix C) zijn getest op de EL-X8 computer van het Mathematisch Centrum aan de hand van 4 series testproblemen (appendix B), waarvan de moeilijkheidsgraad toeneemt van serie A naar serie D.

### 4.1 Rugzakalgoritmen

Tabel 1 geeft aanleiding tot de volgende opmerkingen:

1. De algoritme van Nemhauser is langzamer dan de algoritme van Hu, uitgezonderd voor serie A. De problemen van serie A bezitten een dusdanige dominantie, dat slechts weinig (1-10) variabelen niet gedomineerd worden. Andere hier niet opgenomen testproblemen met $10 \leq n \leq 25$ wijzen eveneens uit dat de beperking van de administratie tot sprongpunten alleen bij kleine n voordeel kan opleveren.

2. De rekentijden van de algoritmen van Hu en Gilmore & Gomory 1a zijn ongeveer gelijk, zoals te verwachten is, omdat beide een zelfde aantal vergelijkingen uitvoeren.

3. Onder de algoritmen, die dynamische programmering gebruiken, is algoritme 4 van Gilmore & Gomory 1b / Shapiro & Wagner superieur. De invloed van de wijzigingen t.o.v. algoritme 3 kan worden afgeleid uit tabel 2. In de C-serie speelt periodiciteit geen rol; alle winst is het gevolg van een kleiner aantal vergelijkingen. Bij een groot rechterlid b telt dit voordeel zwaarder, omdat de turnpike variabele

TABEL 1: REKENTIJDEN VOOR RUGZAKPROBLEMEN (in seconden)

| PROBLEEM | ALGORITME | | | | |
|---|---|---|---|---|---|
| | 1 Hu | 2 Nemhauser | 3 GG 1 a | 4 GG1b/S.W. | 7 B.& B. |
| A - 250 - 19 | .3 | .3 | .3 | .3 | .3 |
| 25 | 2.4 | 2.7 | 2.8 | .4 | .3 |
| 50 | 4.5 | 5.1 | 5.6 | .5 | .3 |
| 500 - 19 | .5 | .5 | .4 | .4 | .4 |
| 25 | 3.4 | 2.0 | 2.8 | .7 | .5 |
| 50 | 6.3 | 3.4 | 5.1 | .9 | .5 |
| 1000 - 19 | .9 | 1.0 | .8 | .8 | .9 |
| 25 | 9.3 | 7.2 | 9.8 | 1.4 | 1.0 |
| 50 | 17.1 | - ** | 18.9 | 1.8 | 1.0 |
| B - 250 - 01 | 4 | 5 | 3 | 2 | 1 |
| 05 | 19 | 32 | 17 | 4 | 2 |
| 500 - 01 | 9 | 14 | 7 | 3 | 2 |
| 05 | 44 | 98 | 42 | 4 | 3 |
| 1000 - 01 | 16 | 28 | 12 | 3 | 4 |
| 05 | 80 | 158 | 66 | 3 | 4 |
| C - 250 - 11 | (55)* | 70 | 30 | 14 | 10 |
| 21 | (160) | 189 | 96 | 24 | 34 |
| 500 - 11 | (220) | (275) | 118 | 52 | 34 |
| 21 | (630) | (750) | (350) | 95 | 190 |
| 1000 - 11 | (900) | (1100) | (450) | 199 | 130 |
| 21 | (2500) | (3000) | (1200) | 377 | 1394 |
| D - 10 - 08 | | | 1.3 | .6 | 14.6 |
| 10 | | | 1.9 | .7 | 3.9 |
| 20 | | | 4.9 | .9 | 171.4 |
| 40 | | | 11.0 | 1.1 | 10.6 |
| 60 | | | 19.0 | 1.4 | .4 |
| 25 - 08 | | | 3.7 | .8 | 173.1 |
| 10 | | | 5.2 | .8 | 1.0 |
| 20 | | | 12.1 | 1.0 | 20.3 |
| 40 | | | 27.3 | 1.2 | 2113.6 |
| 60 | | | 43.9 | 1.4 | >1800 |

* schatting

** overschrijding geheugencapaciteit

TABEL 2. VERGELIJKING ALGORITMEN 3 EN 4

| PROBLEEM | b | R(a) periodiek vanaf | REKENTIJD(sec) | |
|---|---|---|---|---|
| | | | ALG.3 | ALG.4 |
| B-1000-01 | 1016 | 350 | 12 | 3 |
| -05 | 5076 | | 66 | 3 |
| C-1000-11 | 3308 | >11000 | (450) | 199 |
| -21 | 6315 | | (1200) | 377 |
| D- 25-10 | 1010 | 658 | 5 | .8 |
| -60 | 6060 | | 44 | 1.4 |

relatief vaker optimaal is naarmate de rugzakfunctie R(a) het periodieke deel van zijn domein nadert: de verhouding tussen de rekentijden voor de twee C-problemen is 3 bij algoritme 3 en 2 bij algoritme 4. In de B- en D-serie wordt de stijging van de rekentijden in algoritme 3 sterk afgevlakt in algoritme 4 door de vroeg optredende periodiciteit.

4. In de C-serie neemt de relatieve dichtheid langzaam toe bij stijgende $a_j$. Op den duur, bij grote b, zullen de variabelen met grote $a_j$ de voorkeur hebben. Bij de in procedure 4 gehanteerde volgorde naar stijgende $a_j$ moet het grootste deel van de variabelen doorlopen worden volgens functionaalvergelijking (19), als de beslissing i(a) een hoge index heeft. Op voorhand verwacht men dat een algoritme, dat de variabelen omgekeerd nummert, minder vergelijkingen heeft uit te voeren. Het tegendeel bleek echter; de oorzaak hiervan is de kleine waarde van b t.o.v. de grootste $a_j$ in de C-serie, waardoor de situatie, dat variabelen met grote $a_j$ optimaal zijn, zelden kan voorkomen. Daarnaast heeft omkering van de volgorde het bezwaar, dat de contrôle op dominantie niet meer in de algoritme past en dat lemma 5 niet toegepast kan worden.
Hoewel functionaalvergelijking (19) een willekeurige permutatie van de variabelen, bv. naar dalende relatieve dichtheden, toestaat, hebben de twee bovengenoemde permutaties het voordeel, dat, zodra bij één vergelijking geconstateerd is dat b overschreden wordt, de overige vergelijkingen nagelaten kunnen worden.

5. De rekentijden van algoritme 4 en branch-and-bound algoritme 7 ontlopen elkaar niet veel voor de series A,B en C. Wanneer algoritme 4 sneller is betreft het problemen, waarin het bovengrensmechanisme slecht functioneert:
   - In serie C zijn de verschillen tussen de relatieve dichtheden van de variabelen uiterst gering: iedere deelverzameling heeft nagenoeg dezelfde bovengrens; de diepte van de boom van gesplitste deelverzamelingen blijft gelijk, bij stijgende b, maar de breedte neemt toe en daarom ook de rekentijd.
   - In serie D komen geen variabelen met kleine $a_j$ voor ($a_j > 100$). Bij sommige waarden van b zijn de combinaties van relatief hoogwaardige variabelen slecht passend, terwijl de resterende slack niet kan worden opgevuld met kleine $a_j$. Bijgevolg ligt de waarde van de pretendent LB ver af van UB(1) en moet de aftelling lang worden voortgezet. Voor een dergelijk soort problemen fluctueren de rekentijden van de branch-and-bound algoritme sterk, als b een interval doorloopt.

   Voor de gemakkelijker problemen van serie A en B heeft de branch-and-bound algoritme 7 de prettige eigenschap dat de rekentijden niet van b afhangen. De rekentijden van algoritme 4 nemen regelmatig toe bij stijgende b, zolang R(b) nog niet periodiek is, om daarna practisch constant te blijven (afgezien van een lichte stijging voor initialisering van r(a), $0 \leq a \leq b$).
   Een ander vergelijkingspunt is de benodigde geheugenruimte.
   Nadeel van de dynamische programmeringsmethode is dat veel geheugenruimte nodig is: ook bij een andere programmering van de algoritme zijn minimaal vereist 2 arrays, met een lengte gelijk aan de grootste $a_j$, in het kerngeheugen, plus 1 array met een lengte, gelijk aan het niet-periodieke interval van R(a), in een extern geheugen. De branch-and-bound algoritme heeft dit bezwaar niet.

## 4.2 Knapzakalgoritmen

Tabel 3 (zie blz.26 ) geeft aanleiding tot de volgende opmerkingen:

1. De tijden voor de algoritmen van Hu en Gerhardt zijn ongeveer gelijk. In alle hier opgenomen testproblemen behoeft Gerhardt slechts één knapzakprobleem op te lossen, voor enkele andere problemen moesten twee knapzakproblemen worden opgelost, maar i.h.a. schijnt de prijs voor het geringer gebruik van geheugenruimte (vgl.probleem C-250-05) zelden betaald te hoeven worden. Men moet evenwel bedenken dat in de optimale oplossingen van de series B en C slechts enkele variabelen positief zijn.

2. De algoritme van Greenberg en Hegerich is voor de geteste problemen geen verbetering t.o.v. algoritme 8; de oorzaak hiervan ligt in de sterke toename van de administratieve bezigheden.

3. Voor knapzakproblemen is de branch-and-bound algoritme 8 te prefereren, uitgezonderd voor problemen, waarbij kleine $a_j$ ontbreken (serie D). Voor dergelijke problemen falen in feite alle algoritmen bij een groter aantal variabelen.
   Wat betreft het gedrag van de rekentijden van algoritme 8, afhankelijk van de grootte van b, kunnen dezelfde opmerkingen gemaakt worden als voor de branch-and-bound rugzakalgoritme.

## 5. CONCLUSIES

Zowel voor het rugzakprobleem als voor het knapzakprobleem kan niet één algoritme als uniform het beste voor alle soorten problemen beschouwd worden.
Wanneer naast de bijvoorwaarde $\sum a_j x_j \leq b$ andere voorwaarden vervuld moeten worden, komen branch-and-bound algoritmen het eerst in aanmerking, enerzijds omdat deze het eenvoudigst aan het probleem zijn aan te passen, anderzijds, omdat de eigenschappen, waarop goed werkende rugzakalgoritmen m.b.v. dynamische programmering zijn gebaseerd, dan i.h.a. niet meer gelden.
Voor knapzakproblemen, waarin geen variabelen met kleine $a_j$ voorkomen, werkt geen van de behandelde algoritmen bevredigend.

TABEL 3: REKENTIJDEN VOOR KNAPZAKPROBLEMEN (in seconden)

| PROBLEEM | ALGORITME | | | |
|---|---|---|---|---|
| | 5 Hu | 6 Gerhardt | 8 B. & B. | 9 B.& B.(GH) |
| A - 250 - 19 | 2 | 2 | 1.3 | 15 |
| 25 | 70 | 60 | 2.0 | 45 |
| 50 | 114 | 99 | 2.3 | 10 |
| 500 - 19 | 3 | 3 | 2.5 | 43 |
| 25 | 275 | 235 | 4.9 | 33 |
| 50 | 454 | 390 | 4.6 | 12 |
| 1000 - 19 | 8 | 7 | 6.7 | 953 |
| 25 | > 800 | 927 | 9.0 | 513 |
| 50 | | | 9.4 | |
| B - 250 - 01 | 12 | 12 | 2 | 223 |
| 05 | 110 | 97 | 2 | 263 |
| 500 - 01 | | 44 | 5 | |
| 05 | | | 5 | |
| 1000 - 01 | | | 7 | |
| 05 | | | 8 | |
| C - 250 - 01 | 49 | 47 | 9 | |
| 05 | ** | 114 | 29 | |
| 500 - 01 | | 187 | 34 | |
| 05 | | | 162 | |
| 1000 - 01 | | | 125 | |
| 05 | | | 1165 | |
| D - 25 - 08 | 5.2 | 5.3 | 34.6 | > 500 |
| 10 | 6.4 | 6.5 | 150.4 | |
| 20 | 10.8 | 11.1 | 49.3 | |
| 40 * | 12.4 | 13.0 | .2 | |
| 60 * | 12.9 | 13.0 | .2 | |

* probleem triviaal

** overschrijding geheugencapaciteit

APPENDIX A: Dominantie

Bij rugzakproblemen zijn soms variabelen aan te wijzen, die in geen enkele oplossing van RP(b) voorkomen, bv. $x_j$ als $a_j = a_k$ en $c_j < c_k$. We nemen aan dat onder de variabelen geen duplicaten voorkomen, d.w.z. $a_j \neq a_k$ of $c_j \neq c_k$ als $j \neq k$, en dat $0 < a_j \leq b$ voor $j=1,\ldots,n$. Algemeen definiëren we:

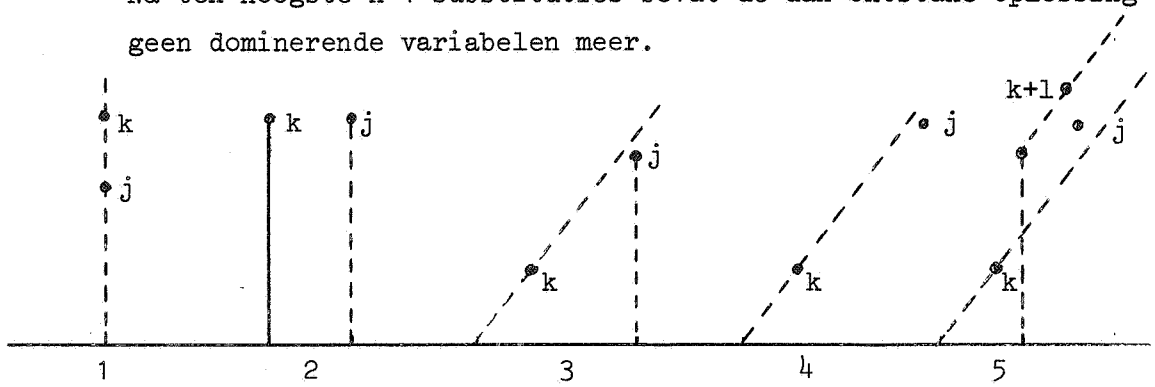
Def.: Een variabele $x_j$ wordt in een gegeven rugzakprobleem gedomineerd als $c_j \leq R_{-j}(a_j)$.

Stelling: R(b) kan worden gerealiseerd door een oplossing zonder gedomineerde variabelen.

Bewijs: Stel $x_{max}$ is de grootste gedomineerde variabele in oplossing $x_0$ die R(a) realiseert, d.w.z.

$$a_{max} = \max \{ a_j \mid x_j > 0,\ c_j \leq R_{-j}(a_j) \}.$$

$c_{max} < R_{-max}(a_{max})$ is strijdig met de optimaliteit van x. Omdat geen duplicaten voorkomen, is dan de grootste variabele die positief is in een oplossing y die $R_{-max}(a_{max})$ realiseert, kleiner dan $x_{max}$. Na substitutie van $x_{max}$ maal deze oplossing y, geldt voor de nieuwe oplossing x' van RP(b), óf $x'_{max}$ bestaat niet, óf $a'_{max} < a_{max}$.
Na ten hoogste n-1 substituties bevat de dan ontstane oplossing geen dominerende variabelen meer.




Een variabele $x_j$ wordt bv. gedomineerd door $x_k$ (en $x_l$) als:

1) $a_j = a_k$ en $c_j < c_k$ ;
2) $c_j \leq \max \{ c_k \mid k \neq j,\ a_k < a_j \}$ ;
3) $a_j = p\, a_k$ en $c_j \leq p\, c_k$ ($p \geq 2$ en geheel) ;
4) $c_j \leq \max \{ pc_k \mid k \neq j,\ pa_k \leq a_j,\ p \geq 2 \text{ en geheel} \}$ ;
5) $c_j \leq \max \{ c_l + pc_k \mid k \neq j,\ l \neq j,\ a_l + pa_k \leq a_j,\ p \geq 1 \text{ en geheel} \}$.

In sommige algoritmen (b.v. rugzakalgoritmen van Gilmore & Gomory) is dominantie tijdens de uitvoering van de algoritme gemakkelijk vast te stellen; in andere is dat niet mogelijk en moet men zich ertoe beperken om vooraf eenvoudige vormen van dominantie na te gaan (branch-and-bound rugzakalgoritme, rugzakalgoritme van Hu) en eventueel ingewikkelder vormen tijdens de uitvoering (rugzakalgoritmen van Hu).
De voor deze controles benodigde tijd is meestal gering in verhouding tot de totale rekentijd of de verkregen tijdwinst.

APPENDIX B: Testproblemen

Bij het testen van de procedures van appendix C zijn 4 series testproblemen gebruikt. Een probleem wordt aangeduid door probleemnummer E-n-p:

- de letter E identificeert de serie;
- n is het aantal variabelen van het probleem;
- p is een getal dat problemen met verschillend rechterlid in dezelfde serie E en met hetzelfde aantal variabelen n van elkaar onderscheidt.

Tabel 4 beschrijft de manier waarop de problemen gegenereerd zijn en geeft enige karakteristieken van de series. (zie app.B 2)

Tabel 5 geeft de grootte van het rechterlid b van de bijvoorwaarde, waarboven de rugzakfunctie R(b) periodiek is, zoals bepaald door algoritme 4 van appendix C.

TABEL 5: PERIODICITEIT VAN RUGZAKFUNCTIES

| n \ SERIE | A | B | C | D |
|---|---|---|---|---|
| 10 | | | | 927 |
| 25 | | | | 658 |
| 250 | 19 | 815 | 2289 | |
| 500 | 29 | 970 | 5963 | |
| 1000 | 26 | 350 | >11000 | |

TABEL 4: KARAKTERISTIEKEN VAN TESTPROBLEMEN

| | Serie A | Serie B | Serie C | Serie D |
|---|---|---|---|---|
| n<br>p | 250-500-1000<br>19-25-50 | 250-500-1000<br>01-05 | 250-500-1000<br>11-21 | 10-25<br>08-10-20-40-60 |
| $a_j$ | $< 5 + 20t^{*} >^{\Delta}$ | $j + 15$ | $\begin{cases} 3j+8, & j \text{ oneven} \\ 3j+7, & j \text{ even} \end{cases}$ | $100 + j$ |
| $c_j$ | $< 500+2000t >/100$ | $\dfrac{< a_j(100+f_n^{\nabla}(t-.5)) >}{100}$ | $< 100a_j^{1.001} >/100$ | $100 + 2j$ |
| b | $\begin{cases} 19 & p=19 \\ 5[3np/100]+4. & p\neq 19 \end{cases}$ | $pa_n + 1$ | $< pa_n/10 >$ | $pa_1$ |
| $\min_j a_j$ | 5 | 16 | 11 | 101 |
| $a_j$ | dicht opeen,<br>deels gelijk | verschillend | verschillend, met grotere intervallen dan serie B | verschillend |
| $c_j/a_j$ | zeer sterk variërend | variërend | $a_j^{.001}$ | stijgend van 1 tot $1+n/(100+n)$ |
| $b/\max_j a_j$ | $\begin{cases} 1 & , p=19 \\ .006np, & p\neq 19 \end{cases}$ | p | p/10 | $p/(1+.01n)$ |
| dominantie | zeer sterk: 1 à 2 %<br>niet-gedomineerde<br>variabelen | sterk: $\pm$ 30<br>niet-gedomineerde<br>variabelen | geen | geen |

* t is de waarde van een aselecte trekking uit een homogene 0-1 verdeling

$\Delta$ $< y > = [y+.5]$

$\nabla$ $f_n$ is zo bepaald dat in elk probleem van serie B $\pm$ 30 niet-gedomineerde variabelen voorkomen

APPENDIX C: ALGOL-procedures

1. Rugzakalgoritme van Hu
2. Rugzakalgoritme van Nemhauser
3. Rugzakalgoritme van Gilmore & Gomory 1A
4. Rugzakalgoritme van Gilmore & Gomory 1B / Shapiro & Wagner
5. Knapzakalgoritme van Hu
6. Knapzakalgoritme van Gerhardt
7. Branch-and-bound rugzakalgoritme
8. Branch-and-bound knapzakalgoritme
9. Branch-and-bound knapzakalgoritme van Greenberg & Hegerich
10. Branch-and-bound algoritme voor rugzakproblemen met bovengrenzen

```
57
58
59
60
61
62
63
64
65          REAL PROCEDURE RUCKSACKDP(N,C,A,B,X);
66          VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
67          BEGIN   INTEGER J, K, AJ, Y, AMAX; REAL CJ, RY;
68                  INTEGER ARRAY P[1:N], I[0:B]; ARRAY R[0:B], Q[1:N];
69
70                  PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
71                  ITER:   IF UP>LO+1 THEN
72                          BEGIN   INTEGER M; J:= P[LO]; CJ:= Q[J]; M:= UP;
73                                  FOR K:= LO + 1 STEP 1 UNTIL M DO IF Q[P[K]] ≤ CJ THEN
74                                  BEGIN   FOR M:= M STEP -1 UNTIL K DO IF Q[P[M]] ≥ CJ THEN
75                                          BEGIN Y:= P[K]; P[K]:= P[M]; P[M]:= Y; M:= M - 1; GOTO NEXTK END;
76                                          M:= K - 1; GOTO IN;
77                          NEXTK:  END;
78                          IN:     P[LO]:= P[M]; P[M]:= J; IF M - LO > UP - M THEN
79                                  BEGIN SORTP(M + 1,UP); UP:= M - 1 END       ELSE
80                                  BEGIN SORTP(LO,M - 1); LO:= M + 1 END;      GOTO ITER
81                          END             ELSE
82                          IF UP>LO        THEN
83                          BEGIN   K:= P[LO]; J:= P[UP];
84                                  IF Q[K] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= K END
85                          END
86                  SORTP;
87
88                  FOR Y:= B STEP -1 UNTIL 0 DO R[Y]:= 0; AMAX:= 0;
89                  FOR J:= N STEP -1 UNTIL 1 DO IF A[J] > B THEN X[J]:= 0 ELSE
90                  BEGIN   CJ:= C[J]; AJ:= A[J]; X[J]:= 0; IF AJ>AMAX THEN AMAX:= AJ;
91                          IF R[AJ]<CJ THEN BEGIN R[AJ]:= CJ; I[AJ]:= J END
92                  END;
93                  CJ:= 0; N:= 0;
94                  FOR AJ:= 1 STEP 1 UNTIL AMAX DO IF R[AJ] > CJ THEN
95                  BEGIN   CJ:= R[AJ]; IF I[AJ]>0 THEN
96                          BEGIN   N:= N+1; J:= P[N]:= I[AJ]; Q[J]:=CJ/AJ; R[AJ]:= 0;
97                                  Y:= AJ; RY:= CJ+CJ; FOR Y:= Y+AJ WHILE Y≤AMAX DO
98                                  IF R[Y]>RY THEN RY:= R[Y]+CJ ELSE
99                                  BEGIN R[Y]:= RY; RY:= RY+CJ; I[Y]:= 0 END
100                         END
101                 END;
102                 SORTP(1,N);
103                 FOR Y:= AMAX STEP -1 UNTIL 0 DO R[Y]:= 0;
104                 FOR K:= 1 STEP 1 UNTIL N DO
105                 BEGIN   J:= P[K]; CJ:= C[J]; AJ:= A[J];
106                         IF R[AJ] < CJ THEN FOR Y:= AJ STEP 1 UNTIL B DO
107                         IF R[Y - AJ] + CJ > R[Y] THEN BEGIN R[Y]:= R[Y - AJ] + CJ; I[Y]:= J END
108                 END;
109                 RUCKSACKDP:= R[B];
110                 FOR Y:= 0, Y + 1 WHILE R[Y] = 0 DO I[Y]:= 0;
111                 FOR J:= I[B] WHILE J > 0 DO BEGIN X[J]:= X[J] + 1; B:= B - A[J] END
112         END RUCKSACKDP;
113
114
115
116
```

```
117
118
119
120         REAL PROCEDURE RUCKSACKDPNEM(N,C,A,B,X);
121         VALUE N,B; INTEGER N,B; ARRAY C; INTEGER ARRAY A,X;
122         BEGIN   INTEGER AJ,AMAX,NEXTA,Y,INDEX,J,K,NI,B1; REAL CJ,RY,INF,INF2;
123                 INTEGER ARRAY P[1:N],I,NEXT[0:B+1]; ARRAY R[0:B+1],Q[1:N];
124
125                 PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
126                 ITER:   IF UP>LO+1 THEN
127                         BEGIN   INTEGER M; J:= P[LO]; CJ:= Q[J]; M:= UP;
128                                 FOR K:= LO + 1 STEP 1 UNTIL M DO IF Q[P[K]] ≤ CJ THEN
129                                 BEGIN   FOR M:= M STEP -1 UNTIL K DO IF Q[P[M]] ≥ CJ THEN
130                                         BEGIN Y:= P[K]; P[K]:= P[M]; P[M]:= Y; M:= M - 1; GOTO NEXTK END;
131                                         M:= K - 1; GOTO IN;
132                                 NEXTK:  END;
133                                 IN:     P[LO]:= P[M]; P[M]:= J; IF M - LO > UP - M THEN
134                                         BEGIN SORTP(M + 1,UP); UP:= M - 1 END       ELSE
135                                         BEGIN SORTP(LO,M - 1); LO:= M + 1 END;      GOTO ITER
136                                 END             ELSE
137                                 IF UP>LO        THEN
138                                 BEGIN   K:= P[LO]; J:= P[UP];
139                                         IF Q[K] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= K END
140                                 END
141                 SORTP;
142
143                 AMAX:= -1; FOR J:= N STEP -1 UNTIL 1 DO IF A[J]>B THEN X[J]:= 0 ELSE
144                 BEGIN   CJ:= C[J]; AJ:= A[J]; X[J]:= 0; IF AJ>AMAX THEN
145                         BEGIN FOR AMAX:= AMAX+1 STEP 1 UNTIL AJ DO R[AMAX]:= 0; AMAX:= AJ END;
146                         IF R[AJ]<CJ THEN BEGIN R[AJ]:= CJ; I[AJ]:= J END
147                 END;
148                 CJ:= 0; NI:= 0;
149                 FOR AJ:= 1 STEP 1 UNTIL AMAX DO IF R[AJ]>CJ THEN
150                 BEGIN   CJ:= R[AJ]; IF I[AJ]>0 THEN
151                         BEGIN   NI:= N+1; J:= P[N]:= I[AJ]; Q[J]:=CJ/AJ; R[AJ]:= 0;
152                                 Y:= AJ; RY:= CJ+CJ; FOR Y:= Y+AJ WHILE Y≤AMAX DO
153                                 IF R[Y]>RY THEN RY:= R[Y]+CJ ELSE
154                                 BEGIN R[Y]:= RY; RY:= RY+CJ; I[Y]:= 0 END
155                         END
156                 END;
157                 SORTP(1,N); I[0]:= 0; B1:= B+1; NEXT[0]:= B1; R[B1]:= INF:= ⎵600; R[0]:= 0; INF2:= ⎵500;
158                 FOR K:= 1 STEP 1 UNTIL N DO
159                 BEGIN   J:= P[K]; AJ:= A[J]; CJ:= C[J];
160                         IF R[AJ]≥CJ THEN GOTO NEM7; INDEX:= 0; NEXTA:= NEXT[0]; Y:= AJ;
161                 NEM1:   RY:= CJ+R[Y-AJ]; IF RY<R[NEXTA] THEN GOTO IF Y<NEXTA THEN NEM3 ELSE NEM6;
162                 NEM2:   IF NEXTA<Y THEN INDEX:= NEXTA; NEXTA:= NEXT[NEXTA];
163                         IF R[NEXTA]≤RY THEN GOTO NEM2; IF Y≥NEXTA THEN GOTO NEM6;
164                 NEM3:   NI:= NEXT[INDEX]; IF NI>Y THEN GOTO NEM5;
165                 NEM4:   NI:= NEXT[NI]; IF NI≤Y THEN GOTO NEM4;
166                 NEM5:   NEXT[Y]:= NI; NEXT[INDEX]:= INDEX:= Y; R[Y]:= RY; I[Y]:= J;
167                 NEM6:   Y:= AJ+NEXT[Y-AJ]; IF Y≤B THEN GOTO NEM1;
168                         IF R[NEXTA]≤INF2 THEN
169                         FOR NEXTA:= NEXTA,NEXT[NEXTA] WHILE R[NEXTA]≤INF2 DO INDEX:= NEXTA; NEXT[INDEX]:= B1;
170                 NEM7:
171                 END J;
172                 RUCKSACKDPNEM:= R[INDEX];
173                 FOR J:= I[INDEX] WHILE J>0 DO BEGIN X[J]:= X[J]+1; INDEX:= INDEX-A[J] END
174         END RUCKSACKDPNEM;
175
176
```

```
177
178
179
180
181
182
183
184
185
186         REAL PROCEDURE RUCKSACKDPGG1A(N,C,A,B,X);
187         VALUE N,B; INTEGER N,B; ARRAY C; INTEGER ARRAY A,X;
188         BEGIN   INTEGER Y,AJ,AMAX,BMAX,ARC,J,N1; REAL CJ,RMAX;
189                 INTEGER ARRAY P[1:N],AP[1:N+1],I[0:B]; ARRAY CP[1:N],R[0:B];
190
191                 FOR Y:= B STEP -1 UNTIL 0 DO R[Y]:= 0; RMAX:= 0; N1:= 1;
192                 FOR J:= N STEP -1 UNTIL 1 DO
193                 BEGIN   CJ:= C[J]; AJ:= A[J]; X[J]:= 0;
194                         IF IF AJ≤B THEN CJ>R[AJ] ELSE FALSE THEN
195                         BEGIN R[AJ]:= CJ; I[AJ]:= J+N END
196                 END;
197
198                 FOR Y:= 1 STEP 1 UNTIL B DO
199                 IF R[Y]≤RMAX THEN I[Y]:= 0 ELSE
200                 BEGIN   RMAX:= R[Y]; IF I[Y]>N THEN
201                         BEGIN P[N1]:= I[Y]-N; I[Y]:= N1; AP[N1]:= Y; CP[N1]:= RMAX; N1:= N1+1; AP[N1]:= B END;
202                         AMAX:= Y+Y; IF AMAX>B THEN AMAX:= B; J:= 1;
203                         FOR ARC:= Y+AP[J] WHILE ARC≤AMAX DO
204                         IF RMAX+CP[J]≤R[ARC] THEN J:= J+1 ELSE
205                         BEGIN R[ARC]:= RMAX+CP[J]; I[ARC]:= J; J:= J+1 END
206                 END;
207
208                 FOR J:= I[B] WHILE J=0 DO B:= B-1; RUCKSACKDPGG1A:= R[B];
209                 FOR B:= B,B-A[J] WHILE B>0 DO BEGIN J:= P[I[B]]; X[J]:= X[J]+1 END
210         END RUCKSACKDPGG1A;
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
```

```
237
238
239
240
241
242
243
244     REAL PROCEDURE RUCKSACKDPGG1BSW(N,C,A,B,X);
245     VALUE N,B; INTEGER N,B; ARRAY C; INTEGER ARRAY A,X;
246     BEGIN   INTEGER Y,AJ,AOPT,BMAX,ARC,J,IY,I1,N1,AP[1]; REAL CJ,COPT,RMAX;
247             INTEGER ARRAY P[1:N+1],AP[0:N+1],I[0:B]; ARRAY CP[1:N],R[0:B];
248             FOR Y:= B STEP -1 UNTIL 0 DO BEGIN R[Y]:= 0; I[Y]:= 0 END;
249             COPT:= RMAX:= 0; AOPT:= BMAX:= 0;
250             FOR J:= N STEP -1 UNTIL 1 DO
251             BEGIN   CJ:= C[J]; AJ:= A[J]; X[J]:= 0; IF IF AJ≤B THEN CJ>R[AJ] ELSE FALSE THEN
252                     BEGIN   R[AJ]:= CJ; I[AJ]:= J+N; CJ:= CJ/AJ; IF CJ>COPT THEN
253                             BEGIN IF AOPT>BMAX THEN BMAX:= AOPT; COPT:= CJ; AOPT:= AJ END ELSE
254                             IF AJ>BMAX THEN BMAX:= AJ
255                     END
256             END;
257             COPT:= R[AOPT]; P[1]:= I[AOPT]-N; N1:= I[AOPT]:= 1; I[0]:= N+1;
258
259             FOR Y:= 1 STEP 1 UNTIL BMAX DO
260             IF R[Y]≤RMAX THEN I[Y]:= 0 ELSE
261             BEGIN   RMAX:= R[Y]; IY:= I[Y]; ARC:= Y+AOPT;
262                     IF IF ARC≤B THEN RMAX+COPT≥R[ARC] ELSE FALSE THEN
263                     BEGIN   R[ARC]:= RMAX+COPT; I[ARC]:= 1; IF ARC=BMAX THEN
264                             FOR I1:= 1, I[BMAX] WHILE I1≤1 DO BMAX:= BMAX-1
265                     END;
266                     IF IY≤1 THEN GOTO NEXTY; IF IY>N THEN
267                     BEGIN N1:= N1+1; P[N1]:= IY-N; IY:= I[Y]:= N1; AP[N1]:= Y; CP[N1]:= RMAX END;
268                     AJ:= AP[IY]; I1:= IY+1; AP[1]:= AP[I1]; AP[I1]:= B; J:= 2;
269                     FOR ARC:= Y+AP[J] WHILE ARC≤B DO
270                     IF RMAX+CP[J]≤R[ARC] THEN J:= J+1 ELSE IF I[ARC-AJ]=J THEN
271                     BEGIN R[ARC]:= RMAX+CP[J]; I[ARC]:= I1:= J; J:= J+1 END ELSE J:= J+1;
272                     IF I1≤IY THEN BEGIN ARC:= Y+AP[I1]; IF ARC>BMAX THEN BMAX:= ARC; I1:= IY+1 END;
273                     AP[I1]:= AP[1];
274             NEXTY:
275             END;
276
277     BACK:   IF BMAX+AOPT≥B THEN COPT:= 0 ELSE
278             BEGIN X[P[1]]:= IY:= (B-BMAX)÷AOPT; B:= B-IY*AOPT; COPT:= IY*COPT END;
279             FOR IY:= I[B] WHILE IY=0 DO B:= B-1; RUCKSACKDPGG1BSW:= COPT + R[B];
280             IF B>0 THEN FOR B:= B,B-A[J] WHILE B>0 DO BEGIN J:= P[I[B]]; X[J]:= X[J]+1 END
281     END RUCKSACKDPGG1BSW;
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
```

```
297
298
299
300
301         REAL PROCEDURE KNAPSACKDP(N,C,A,B,X);
302         VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
303         BEGIN   REAL CJ,KY; INTEGER AJ,Y,J,JU,BMAX,M; BOOLEAN NEW;
304                 INTEGER ARRAY P,BACK[1:N],I[1:AVAILABLE-1000-(IF B≥N THEN 2*(B+N) ELSE 4*N)];
305                 ARRAY K[-1:IF B≥N THEN B ELSE N];
306
307                 PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
308                 ITER:   IF UP>LO+1 THEN
309                         BEGIN   INTEGER M; Y:= P[LO]; KY:= K[Y]; M:= UP;
310                                 FOR J:= LO+1 STEP 1 UNTIL M DO IF K[P[J]]≤KY THEN
311                                 BEGIN   FOR M:= M STEP -1 UNTIL J DO IF K[P[M]]≥KY THEN
312                                         BEGIN J0:= P[J]; P[J]:= P[M]; P[M]:= J0; M:= M-1; GOTO NEXTJ END;
313                                         M:= J-1; GOTO IN;
314                                 NEXTJ:  END;
315                                 IN:     P[LO]:= P[M]; P[M]:= Y; IF M-LO>UP-M THEN
316                                         BEGIN SORTP(M+1,UP); UP:= M-1 END    ELSE
317                                         BEGIN SORTP(LO,M-1); LO:= M+1 END;   GOTO ITER
318                                 END     ELSE
319                                 IF UP>LO THEN
320                                 BEGIN   Y:= P[LO]; J:= P[UP]; IF K[Y]<K[J] THEN
321                                         BEGIN P[LO]:= J; P[UP]:= Y END
322                                 END
323                 SORTP;
324
325                 J:= N; N:= 0; FOR J:= J STEP -1 UNTIL 1 DO
326                 BEGIN   AJ:= A[J]; CJ:= C[J]; X[J]:= 0; IF AJ≤B ∧ CJ>0 THEN
327                         BEGIN N:= N+1; P[N]:= J; K[J]:= CJ/AJ END
328                 END;
329                 SORTP(1,N); M:= 1; BMAX:= 0; K[0]:= 0;
330
331                 FOR J0:= 1 STEP 1 UNTIL N DO
332                 BEGIN   J:= P[J0]; CJ:= C[J]; AJ:= A[J];
333                         BACK[J]:= M; NEW:= BMAX<B; IF NEW THEN
334                         BEGIN   KY:= K[BMAX]; Y:= BMAX; BMAX:= BMAX+AJ; NEW:= BMAX<B;
335                                 IF NEW THEN BEGIN I[M]:= B; M:= M+1 END ELSE BMAX:= B;
336                                 FOR Y:= Y+1 STEP 1 UNTIL BMAX DO K[Y]:= KY
337                         END;
338                         Y:= BMAX; FOR KY:= K[Y-AJ]+CJ WHILE Y≥AJ DO IF KY>K[Y] THEN
339                         BEGIN   K[Y]:= KY; IF ¬NEW THEN
340                                 BEGIN I[M]:= Y; M:= M+1; NEW:= TRUE END; Y:= Y-1
341                         END ELSE IF ¬NEW THEN Y:= Y-1 ELSE
342                         BEGIN I[M]:= Y; M:= M+1; NEW:= FALSE; Y:= Y-1 END;
343                         I[M]:= 0; M:= M+1
344                 END;
345
346                 IF BMAX<B THEN B:= BMAX; KNAPSACKDP:= K[B];
347                 FOR J0:= N STEP -1 UNTIL 1 DO
348                 BEGIN   J:= P[J0]; IF A[J]>B THEN GOTO OUTJ; M:= AJ:= BACK[J];
349                         FOR Y:= I[M] WHILE Y≥B DO M:= M+1;
350                         IF EVEN(M-AJ)=-1 THEN BEGIN X[J]:= 1; B:= B-A[J] END;
351                 OUTJ:
352                 END
353         END KNAPSACKDP;
354
355
356
```

```
357
358
359
360
361
362
363
364         REAL PROCEDURE KNAPSACKDPGER(N,C,A,B,X);
365         VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
366         BEGIN   REAL CJ; INTEGER AJ,J,J0;
367                 INTEGER ARRAY P[1:N],I[0:B]; ARRAY K[-1:IF B≥N THEN B ELSE N];
368
369                 PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
370                 ITER:   IF UP>LO+1 THEN
371                         BEGIN   INTEGER M; J:= P[LO]; CJ:= K[J]; M:= UP;
372                                 FOR J0:= LO+1 STEP 1 UNTIL M DO IF K[P[J0]]≤CJ THEN
373                                 BEGIN   FOR M:= M STEP -1 UNTIL J0 DO IF K[P[M]]≥CJ THEN
374                                         BEGIN AJ:= P[J0]; P[J0]:= P[M]; P[M]:= AJ; M:= M-1; GOTO NEXTJ0 END;
375                                         M:= J0-1; GOTO IN;
376                                 NEXTJ0: END;
377                                 IN:     P[LO]:= P[M]; P[M]:= J; IF M-LO>UP-M THEN
378                                         BEGIN SORTP(M+1,UP); UP:= M-1 END    ELSE
379                                         BEGIN SORTP(LO,M-1); LO:= M+1 END;   GOTO ITER
380                                 END     ELSE
381                                 IF UP>LO THEN
382                                 BEGIN   J0:= P[LO]; J:= P[UP]; IF K[J0]<K[J] THEN
383                                         BEGIN P[LO]:= J; P[UP]:= J0 END
384                                 END
385                 SORTP;
386
387                 INTEGER PROCEDURE KNAP(N); VALUE N; INTEGER N;
388                 BEGIN   INTEGER Y,J,J0,BMAX; REAL KY;
389                         BMAX:= 0; K[0]:= 0;
390                         FOR J0:= 1 STEP 1 UNTIL N DO
391                         BEGIN   J:= P[J0]; CJ:= C[J]; AJ:= A[J]; IF BMAX<B THEN
392                                 BEGIN   KY:= K[BMAX]; J:= I[BMAX]; Y:= BMAX;
393                                         BMAX:= BMAX+AJ; IF BMAX>B THEN BMAX:= B;
394                                         FOR Y:= Y+1 STEP 1 UNTIL BMAX DO BEGIN K[Y]:= KY; I[Y]:= J END
395                                 END;
396                                 Y:= BMAX; IF Y≥AJ THEN
397                                 FOR KY:= K[Y-AJ]+CJ WHILE Y≥AJ DO
398                                 IF KY≤K[Y] THEN Y:= Y-1 ELSE
399                                 BEGIN K[Y]:= KY; I[Y]:= J0; Y:= Y-1 END
400                         END;
401                         IF BMAX<B THEN B:= BMAX; KNAP:= I[B]
402                 END KNAP;
403
404                 J:= N; N:= 0; FOR J:= J STEP -1 UNTIL 1 DO
405                 BEGIN   AJ:= A[J]; CJ:= C[J]; X[J]:= 0; IF AJ≤B ∧ CJ>0 THEN
406                         BEGIN N:= N+1; P[N]:= J; K[J]:= CJ/AJ END
407                 END;
408                 SORTP(1,N);
409
410                 KNAP(N); KNAPSACKDPGER:= K[B]; N:= N+1;
411                 FOR B:= B,B-A[J] WHILE K[B]>0 DO
412                 BEGIN J0:= I[B]; N:= IF J0<N THEN J0 ELSE KNAP(N-1); J:= P[N]; X[J]:= 1 END
413         END KNAPSACKDPGER;
414
415
416
```

```
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73      REAL PROCEDURE RucksackBB(N,C,A,B,X);
74      VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
75      BEGIN   REAL CK, RY; INTEGER J,K,Y,AK,TP,TK; INTEGER ARRAY P[1:N];
76
77              BEGIN   REAL ARRAY Q[1:N], R[1:B]; INTEGER ARRAY I[1:B];
78
79                      PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
80                      ITER:   IF UP > LO + 1 THEN
81                              BEGIN   INTEGER M; K:= P[LO]; CK:= Q[K]; M:= UP;
82                                      FOR J:= LO + 1 STEP 1 UNTIL M DO IF Q[P[J]] ≤ CK THEN
83                                      BEGIN   FOR M:= M STEP -1 UNTIL J DO IF Q[P[M]] ≥ CK THEN
84                                              BEGIN Y:= P[J]; P[J]:= P[M]; P[M]:= Y; M:= M - 1; GOTO NEXTJ END;
85                                              M:= J - 1; GOTO IN;
86                                      NEXTJ:  END;
87                                      IN:     P[LO]:= P[M]; P[M]:= K; IF M - LO > UP - M THEN
88                                              BEGIN SORTP(M + 1,UP); UP:= M - 1 END       ELSE
89                                              BEGIN SORTP(LO,M - 1); LO:= M + 1 END;      GOTO ITER
90                                      END             ELSE
91                                      IF UP > LO      THEN
92                                      BEGIN   K:= P[LO]; J:= P[UP];
93                                              IF Q[K] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= K END
94                                      END
95                      SORTP;
96
97                      TP:= 0; FOR K:= N STEP -1 UNTIL 1 DO IF A[K] > B THEN X[K]:= 0 ELSE
98                      BEGIN   CK:= C[K]; AK:= A[K]; X[K]:= 0;
99                              IF AK>TP  THEN BEGIN FOR TP:= TP+1 STEP 1 UNTIL AK DO R[TP]:= 0; TP:= AK END;
100                             IF R[AK] < CK THEN BEGIN R[AK]:= CK; I[AK]:= K END
101                     END;
102                     CK:= 0; N:= 0;
103                     FOR AK:= 1 STEP 1 UNTIL TP DO IF R[AK] > CK THEN
104                     BEGIN   CK:= R[AK]; IF I[AK] > 0 THEN
105                             BEGIN   N:= N + 1; K:= P[N]:= I[AK]; Q[K]:= CK/AK;
106                                     Y:= AK; RY:= CK + CK; FOR Y:= Y + AK WHILE Y ≤ TP DO
107                                     IF R[Y] > RY THEN RY:= R[Y] + CK ELSE
108                                     BEGIN R[Y]:= RY; RY:= RY + CK; I[Y]:= 0 END
109                             END
110                     END;
111                     SORTP(1,N)
112             END DOMSORT;
113
114             BEGIN   REAL ARRAY CP, QP[1:N]; INTEGER ARRAY AP, MINAP, RP, RK, XPRP, XKRK[1:N];
115
116                     PROCEDURE RUCK;
```

```
                        BEGIN   BOOLEAN POK; REAL CPK, QPL; INTEGER APK, XK, L;
                                CPK:= CP[K]; APK:= AP[K];
                                Y:= B / APK; XK:= IF K = N ∨ B = Y * APK THEN -1 ELSE (B - MINAP[K])/APK - .499999;
                                IF Y > XK THEN
                                BEGIN   RY:= Y * CPK; IF RY > CK THEN
                                        BEGIN   CK:= RY; TP:= TK + 1; RP[TP]:= K; XPRP[TP]:= Y;
                                                FOR J:= TK STEP -1 UNTIL 1 DO BEGIN RP[J]:= RK[J]; XPRP[J]:= XKRK[J] END
                                        END;
                                        IF XK < 0 THEN GOTO OUT
                                END;
                                L:= K + 1; QPL:= QP[L]; POK:= XK > 0;
                                IF POK THEN
                                BEGIN   CK:= CK - XK * CPK; B:= B - XK * APK; TK:= TK + 1; RK[TK]:= K; XKRK[TK]:= XK   END;
                        IN:     IF B * QPL > CK THEN
                                BEGIN   K:= L; FOR J:= K WHILE B < AP[J] DO K:= J + 1;
                                        IF K = L THEN RUCK ELSE IF B * QP[K] > CK THEN RUCK;
                                        IF POK THEN
                                        BEGIN   XK:= XK - 1; POK:= XK > 0;
                                                CK:= CK + CPK; B:= B + APK; IF POK THEN XKRK[TK]:= XK ELSE TK:= TK - 1;
                                                GOTO IN
                                        END
                                END             ELSE
                                IF POK THEN
                                BEGIN   CK:= CK + XK * CPK; B:= B + XK * APK; TK:= TK - 1   END;
                        OUT:
                        END RUCK;

                        Y:= .7; FOR J:= N STEP -1 UNTIL 1 DO
                        BEGIN   K:= P[J]; CP[J]:= CK:= C[K]; AP[J]:= AK:= A[K];
                                QP[J]:= CK/AK; MINAP[J]:= Y; IF AK<Y THEN Y:= AK
                        END;
                        TK:= 0; CK:= 0; K:= 1; RUCK; RUCKSACKBB:= CK;
                        FOR J:= TP STEP -1 UNTIL 1 DO X[P[RP[J]]]:= XPRP[J]
                END BB
        END RUCKSACKBB;
```

```
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193     REAL PROCEDURE KNAPSACKBB(N,C,A,B,X);
194     VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
195     BEGIN   REAL CK,CF,LB; INTEGER J,K,AK,TP,TK; INTEGER ARRAY P[1:N];
196
197             BEGIN   REAL ARRAY Q[1:N];
198
199                     PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
200                     ITER:   IF UP > LO + 1 THEN
201                             BEGIN   INTEGER M; K:= P[LO]; CK:= Q[K]; M:= UP;
202                                     FOR J:= LO + 1 STEP 1 UNTIL M DO IF Q[P[J]] ≤ CK THEN
203                                     BEGIN   FOR M:= M STEP -1 UNTIL J DO IF Q[P[M]] ≥ CK THEN
204                                             BEGIN TP:= P[J]; P[J]:= P[M]; P[M]:= TP; M:= M - 1; GOTO NEXTJ END;
205                                             M:= J - 1; GOTO IN;
206                                     NEXTJ:  END;
207                                     IN:     P[LO]:= P[M]; P[M]:= K; IF M - LO > UP - M THEN
208                                             BEGIN SORTP(M + 1,UP); UP:= M - 1 END       ELSE
209                                             BEGIN SORTP(LO,M - 1); LO:= M + 1 END;      GOTO ITER
210                             END             ELSE
211                             IF UP > LO      THEN
212                             BEGIN   K:= P[LO]; J:= P[UP];
213                                     IF Q[K] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= K END
214                             END
215                     SORTP;
216
217                     K:= N; N:= 0; FOR K:= K STEP -1 UNTIL 1 DO
218                     BEGIN   X[K]:= 0; CK:= C[K]; AK:= A[K]; IF CK > 0 ∧ AK ≤ B THEN
219                             BEGIN N:= N + 1; P[N]:= K; Q[K]:= CK/AK END
220                     END;
221                     SORTP(1,N); J:= N + 1
222             END SORT;
223
224             BEGIN   ARRAY CP[1:N],QP[1:J]; INTEGER ARRAY AP[1:J],XP,XK,MINAP[1:N];
225
226                     PROCEDURE KNAP(BK,CK,F); VALUE BK,CK,F; INTEGER BK,F; REAL CK;
227                     BEGIN   INTEGER APK,L; REAL CPK;
228                             TK:= TK+1; L:= K+1; IF F≥L THEN GOTO BRANCH; L:= L+1;
229                     NEXTK:  FOR APK:= AP[L] WHILE APK>BK DO L:= L+1;
230                             IF CK+BK*QP[L]≤LB THEN GOTO OUT; K:= F:= L;
231                     NEXTF:  FOR APK:= AP[F] WHILE APK≤BK DO
232                             BEGIN BK:= BK-APK; CK:= CK+CP[F]; F:= F+1 END;
233                             CF:= QP[F]*BK; IF CK+CF≤LB THEN GOTO OUT;
234                             L:= L+1; IF CF≤0 THEN
235                             BEGIN   LB:= CK; TP:= TK+F-L;
236                                     FOR J:= TK-1 STEP -1 UNTIL 1 DO XP[J]:= XK[J];
```

```
                                    FOR J:= F-L STEP -1 UNTIL 0 DO XP[TP-J]:= K+J; GOTO OUT
                    END;
BRANCH:             XK[TK]:= K; APK:= AP[K]; CPK:= CP[K]; IF F>L THEN
                    BEGIN   K:= L; KNAP(BK,CK,F); K:= L;
                            BK:= BK+APK; CK:= CK-CPK;
                            GOTO IF CK+BK*QP[F]>LB THEN NEXTP ELSE OUT
                    END;
                    AK:= BK-MINAP[K]; IF AK≥0 THEN KNAP(BK,CK,K) ELSE
                    IF CK≤LB THEN GOTO IF AK+APK≥0 THEN BACK ELSE OUT ELSE
                    BEGIN   FOR J:= TK STEP -1 UNTIL 1 DO XP[J]:= XK[J];
                            LB:= CK; TP:= TK; IF AK+APK<0 THEN GOTO OUT
                    END;
BACK:               BK:= BK+APK; CK:= CK-CPK; GOTO NEXTK;
OUT:                TK:= TK-1
                END KNAP;

                QP[J]:= 0; TP:= AP[J]:= ∞7;
                FOR J:= N STEP -1 UNTIL 1 DO
                BEGIN   K:= P[J]; CP[J]:= CK:= C[K]; AP[J]:= AK:= A[K];
                        QP[J]:= CK/AK; MINAP[J]:= TP; IF AK<TP THEN TP:= AK
                END;
                LB:= 0; K:= -1; TK:= 0;
                IF N=0 THEN TP:= 0 ELSE KNAP(B,0,-1); KNAPSACKBB:= LB;
                FOR J:= TP STEP -1 UNTIL 1 DO X[P[XP[J]]]:= 1
        END
END KNAPSACKBB;
```

## van Greenberg & Hegerich

```
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315   REAL PROCEDURE KNAPSACKBBGH(N,C,A,B,X);
316   VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,X;
317   BEGIN   REAL LB,CJ,QPJ; INTEGER H,J,K,AJ,APJ,BJ,TP; INTEGER ARRAY P[1:N];
318
319           BEGIN   REAL ARRAY Q[1:N];
320
321                   PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
322                   ITER:   IF UP > LO + 1 THEN
323                           BEGIN   INTEGER K; J:= P[LO]; QPJ:= Q[J]; K:= UP;
324                                   FOR H:= LO + 1 STEP 1 UNTIL K DO IF Q[P[H]] ≤ QPJ THEN
325                                   BEGIN   FOR K:= K STEP -1 UNTIL H DO IF Q[P[K]] ≥ QPJ THEN
326                                           BEGIN TP:= P[H]; P[H]:= P[K]; P[K]:= TP; K:= K - 1; GOTO NEXTH END;
327                                           K:= H - 1; GOTO IN;
328                                   NEXTH:  END;
329                                   IN:     P[LO]:= P[K]; P[K]:= J; IF K - LO > UP - K THEN
330                                           BEGIN SORTP(K + 1,UP); UP:= K - 1 END      ELSE
331                                           BEGIN SORTP(LO,K - 1); LO:= K + 1 END;     GOTO ITER
332                                   END             ELSE
333                                   IF UP > LO      THEN
334                                   BEGIN   H:= P[LO]; J:= P[UP];
335                                           IF Q[H] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= H END
336                                   END
337                   SORTP;
338
339                   J:= N; N:= 0; FOR J:= J STEP -1 UNTIL 1 DO
340                   BEGIN   CJ:= C[J]; AJ:= A[J]; IF CJ>0 ∧ AJ≤B THEN
341                           BEGIN N:= N+1; P[N]:= J; Q[J]:= CJ/AJ END ELSE X[J]:= 0
342                   END;
343                   SORTP(1,N)
344           END-SORT;
345
346           BEGIN   INTEGER BFREE; ARRAY CP[0:N],QP[1:N]; INTEGER ARRAY AP,XP,XK,MINAP[0:N];
347
348                   PROCEDURE UPDATE(K,CK); VALUE K,CK; INTEGER K; REAL CK;
349                   BEGIN   FOR TP:= N STEP -1 UNTIL 1 DO XP[TP]:= XK[TP];
350                           TP:= K; LB:= CK+„-7
351                   END;
352
353                   PROCEDURE KNAP(K,BK,CK); VALUE K,BK,CK; INTEGER K,BK; REAL CK;
354                   BEGIN   INTEGER APK; XK[K]:= 0; IF MINAP[K]≤BFREE THEN
355                           BEGIN   BJ:= BK; CJ:= CK;
356                                   FOR J:= K+1 STEP 1 UNTIL N DO IF XK[J]=-1 THEN
```

```
                                        BEGIN   APJ:= AP[J]; IF APJ≥BJ THEN
                                                BEGIN   IF CJ+QP[J]*BJ<LB THEN GOTO RIGHT;
                                                        IF APJ>BJ THEN KNAP(J,BJ,CJ) ELSE UPDATE(J,CJ+CP[J]);
                                                        GOTO RIGHT
                                                END;
                                                BJ:= BJ-APJ; CJ:= CJ+CP[J]
                                        END;
                                        IF CJ>LB THEN UPDATE(N,CJ)
                                ELSE IF CK>LB THEN UPDATE(K-1,CK);
                        RIGHT:  APK:= AP[K]; IF APK>BFREE THEN GOTO OUT; BK:= BK-APK; CK:= CK+CP[K];
                                FOR J:= K-1 STEP -1 UNTIL 1 DO IF XK[J]=-1 THEN
                                BEGIN   BK:= BK+AP[J]; CK:= CK-CP[J]; IF BK≥0 THEN
                                        BEGIN   IF CK+QP[J]*BK<LB THEN GOTO OUT;
                                                IF BK=0 THEN BEGIN UPDATE(J,CK); GOTO OUT END;
                                                BFREE:= BFREE-APK; XK[K]:= 1; KNAP(J,BK,CK);
                                                BFREE:= BFREE+APK; GOTO OUT
                                        END
                                END;
                        OUT:    XK[K]:= -1
                        END KNAP;

                        CP[0]:= LB:= 10-7; AP[0]:= XK[0]:= MINAP[0]:= -1; BJ:= 10↑7; BFREE:= B;
                        FOR H:= N STEP -1 UNTIL 1 DO
                        BEGIN   J:= P[H]; CP[H]:= CJ:= C[J]; AP[H]:= AJ:= A[J]; XK[H]:= -1;
                                QP[H]:= CJ/AJ; MINAP[H]:= BJ; IF AJ<BJ THEN BJ:= AJ
                        END;
                        IF N=0 THEN TP:= 0 ELSE KNAP(0,B,0); KNAPSACKBBGH:= LB-10-7;
                        FOR J:= TP STEP -1 UNTIL 1 DO X[P[J]]:= ABS(XP[J]);
                        FOR J:= TP+1 STEP 1 UNTIL N DO X[P[J]]:= IF XP[J]=1 THEN 1 ELSE 0
                END
        END KNAPSACKBBGH;
```

```
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437     REAL PROCEDURE RUCKSACKUBBB(N,C,A,U,B,X);
438     VALUE N,B; INTEGER N,B; REAL ARRAY C; INTEGER ARRAY A,U,X;
439     BEGIN   REAL CK, RY; INTEGER J, K, AK, TP, TK; INTEGER ARRAY P[1:N];
440
441             BEGIN   REAL ARRAY Q[1:N];
442
443                     PROCEDURE SORTP(LO,UP); VALUE LO,UP; INTEGER LO,UP;
444                     ITER:   IF UP > LO + 1 THEN
445                             BEGIN   INTEGER M; K:= P[LO]; CK:= Q[K]; M:= UP;
446                                     FOR J:= LO + 1 STEP 1 UNTIL M DO IF Q[P[J]] ≤ CK THEN
447                                     BEGIN   FOR M:= M STEP -1 UNTIL J DO IF Q[P[M]] ≥ CK THEN
448                                             BEGIN TP:= P[J]; P[J]:= P[M]; P[M]:= TP; M:= M - 1; GOTO NEXTJ END;
449                                             M:= J - 1; GOTO IN;
450                                     NEXTJ:  END;
451                                     IN:     P[LO]:= P[M]; P[M]:= K; IF M - LO > UP - M THEN
452                                             BEGIN SORTP(M + 1,UP); UP:= M - 1 END       ELSE
453                                             BEGIN SORTP(LO,M - 1); LO:= M + 1 END;      GOTO ITER
454                             END             ELSE
455                             IF UP > LO      THEN
456                             BEGIN   K:= P[LO]; J:= P[UP];
457                                     IF Q[K] < Q[J] THEN BEGIN P[LO]:= J; P[UP]:= K END
458                             END
459                     SORTP;
460
461                     K:= N; N:= 0; FOR K:= K STEP -1 UNTIL 1 DO
462                     BEGIN   X[K]:= 0; CK:= C[K]; AK:= A[K];
463                             IF CK > 0 ∧ AK ≤ B ∧ U[K] ≥ 1 THEN
464                             BEGIN N:= N + 1; P[N]:= K; Q[K]:= CK/AK END
465                     END;
466                     SORTP(1,N); J:= N + 1
467             END SORT;
468
469             BEGIN   REAL ARRAY CP, UPCP[1:N], QP[1:J]; INTEGER ARRAY UPAP[1:J], AP, UP, MINAP, RP, RK, XPRP, XKRK[1:N];
470
471                     PROCEDURE RUCK;
472                     BEGIN   BOOLEAN POK; REAL CPK, CM; INTEGER APK, BM, XK, L, M;
473                             CPK:= CP[K]; APK:= AP[K];
474                             AK:= IF B < UPAP[K] THEN B ↓ APK ELSE UP[K];
475                             XK:= IF K = N ∨ B = AK * APK THEN -1 ELSE (B - MINAP[K])/APK - ,499999;
476                             IF AK > XK THEN
```

```
                        BEGIN   RY:= AK * CPK; IF RY > CK THEN
                                BEGIN   CK:= RY; TP:= TK + 1; RP[TP]:= K; XPRP[TP]:= AK;
                                        FOR J:= TK STEP -1 UNTIL 1 DO BEGIN RP[J]:= RK[J]; XPRP[J]:= XKRK[J] END
                                END;
                                IF XK < 0 THEN GOTO OUT
                        END;
                        IF XK > AK THEN XK:= AK; POK:= XK > 0;
                        IF POK THEN
                        BEGIN   CK:= CK - XK * CPK; B:= B - XK * APK; TK:= TK + 1; RK[TK]:= K; XKRK[TK]:= XK   END;
                        M:= L:= K + 1; CM:= 0; BM:= B;
                IN:     FOR AK:= UPAP[M] WHILE BM ≥ AK DO
                        BEGIN   CM:= CM - UPCP[M]; BM:= BM - AK; M:= M + 1   END;
                        IF BM * QP[M] > CK + CM THEN
                        BEGIN   K:= L;
                                IF B ≥ AP[K] THEN RUCK ELSE
                                BEGIN   K:= K + 1; FOR J:= K WHILE B < AP[J] DO K:= J + 1;
                                        IF B * QP[K] > CK THEN RUCK
                                END;
                                IF POK THEN
                                BEGIN   XK:= XK - 1; POK:= XK > 0;
                                        CK:= CK + CPK; B:= B + APK; IF POK THEN XKRK[TK]:= XK ELSE TK:= TK - 1;
                                        BM:= BM + APK; GOTO IN
                                END
                        END             ELSE
                        IF POK THEN
                        BEGIN   CK:= CK + XK * CPK; B:= B + XK * APK; TK:= TK - 1   END;
                OUT:
                END RUCK;

                QP[J]:= 0; UPAP[J]:= TP:= 47; FOR J:= N STEP -1 UNTIL 1 DO
                BEGIN   K:= P[J]; CP[J]:= CK:= C[K]; AP[J]:= AK:= A[K]; UP[J]:= TK:= U[K];
                        QP[J]:= CK/AK; UPCP[J]:= TK * CK; UPAP[J]:= TK * AK; MINAP[J]:= TP; IF AK < TP THEN TP:= AK
                END;
                CK:= 0; TK:= 0; K:= 1; RUCK; RUCKSACKUBBB:= CK;
                FOR J:= TP STEP -1 UNTIL 1 DO X[P[RP[J]]]:= XPRP[J]
        END BB
END RUCKSACKUBBB;
```

APPENDIX D: Literatuur

1. Bowman, V.J. and G.L. Nemhauser, Deep cuts in integer programming.
CORE Discussion paper, October 1969, Leuven.

2. Bradley, G.H., Transformation of integer programs to knapsack problems.
Discrete Mathematics 1 (1971), 29-45.

3. Burdet, C.-A., Generating all the faces of a polyhedron.
Man. Sc., Res. Rep. no. 271 (1971), Carnegie-Mellon Univ., Pittsburg.

4. Cabot, A., An enumeration algorithm for knapsack problems.
Opns. Res. 18 (1970), 306-311.

5. Geoffrion, A.M. and R.E. Marsten, Integer programming algorithms: a framework and state-of-the-art survey.
Man. Sc., 18 (1972), 465-491.

6. Gerhardt, C., Gedanken zur Lösung des Knapsack-Problems.
Ablauf- und Planungsforschung 11 (1970), 69-83.

7. Gilmore, P.C. and R.E. Gomory, A linear programming approach to the cutting stock problem.
Part I : Opns. Res. 9 (1961), 849-859.
Part II: Opns. Res. 11 (1963), 863-888.

8. Gilmore, P.C. and R.E. Gomory, Multi-stage cutting stock problems of two or more dimensions.
Opns. Res. 13 (1965), 94-120.

9. Gilmore, P.C. and R.E. Gomory, The theory and computation of knapsack functions.
Opns. Res. 14 (1966), 1045-1074.

10. Greenberg, H., An algorithm for the computation of knapsack functions.
Journ. of Math. Anal. and Applications 26 (1969), 159-162.

11. Greenberg, H., and R.L. Hegerich, A branch search algorithm for the knapsack problem.
Man. Sc. 16 (1970), 327-332.

12. Hu, T.C., Integer programming and network flows.
Addison-Wesley, Reading (Mass.) (1969).

13. Kirby, M.J.L., H.R. Love and K. Swarup, A cutting-plane algorithm for extremal point mathematical programming.
Cah. du Centre d'Et. de R.O. 14 (1972), 27-42.

14. Kolesar, P.J., A branch-and-bound algorithm for the knapsack problem.
Man. Sc. 13 (1967), 723-735.

15. Land, A.H. and A.G. Doig, An automatic method of solving discrete programming problems.
Econometrica 28 (1960), 497-520.

16. Lawler, E.L. and D.E. Wood, Branch-and-bound methods: a survey.
Opns. Res. 14 (1966), 699-719.

17. Mitten, L.G., Branch-and-bound methods, general formulation and properties.
Opns. Res. 18 (1970), 24-35.

18. Nemhauser, G.L. and Z. Ullmann, Discrete dynamic programming and capital allocation.
Man. Sc. 15 (1969), 494-505.

19. Shapiro, J.F., Dynamic programming algorithms for the integer programming problem; I: the integer programming problem viewed as a knapsack type problem.
Opns. Res. 16 (1968), 103-121.

20. Shapiro, J.F. and H.M. Wagner, A finite renewal algorithm for the knapsack and turnpike models.
Opns. Res. 15 (1967), 319-341.